E. Floegel



Anwendungsbeispiele

- Datenverwaltungsprogramme
- Geschäftsprogramme
- Programmiertechniken
- Unterhaltungsprogramme
- Künstliche Intelligenz

OK

ISBN 3-88963-200-9

1. Auflage 1984

Gedruckt in der Bundesrepublik Deutschland – Printed in West-Germany – Imprime'en RFA.

E. Floegel

FORTH
Anwendungsbeispiele

Datenverwaltungsprogramme · Geschäftsprogramme
Programmiertechniken · Unterhaltungsprogramme
Künstliche Intelligenz

Vorwort

Das vorliegende Buch soll zeigen, wie man mit der Sprache FORTH Anwendungen programmieren kann. Es werden Beispiele für formatiertes Ausdrucken, für den Aufbau von Datenstrukturen, Rechnungsschreiben und für künstliche Intelligenz gebracht.

In FORTH geschriebene Programme zeigen meist einen persönlichen Programmierstil. Deshalb sollen die vorliegenden Programme hauptsächlich Anregungen zum selbstständigen Programmieren sein.

Das reine Abtippen dürfte nicht die richtige Art sein, das Programmieren in FORTH zu üben. Es ist besser, die Programmidee zu verstehen, und sie dann mit eigenen "Worten" zu programmieren.

Wenn ein gewisser Grundvorrat an definierten Worten vorliegt, dann kann man sehr schnell, wie es in keiner anderen Sprache möglich ist, Programme auf spezielle Anwendungen anpassen.

Dem Leser wünsche ich viel Spaß mit FORTH.

Holzkirchen, Sommer 1984 Ekkehard Flögel

Inhaltsverzeichnis



Einleitung

Die Sprache FORTH wurde in den letzten Jahren benutzt, um einige Anwendungen aus der Praxis zu programmieren. Dabei wurde für die Programmentwicklung ein Wortevorrat entwickelt, der in allen Programmen benutzt wird. Dies sind vor allem Worte, die für den Datenaustausch zwischen Benutzer und Rechner oder Rechner und Peripherie benötigt werden. Ein Großteil des Programmieraufwandes muß in diese Programmteile gesteckt werden, um den Umgang mit einem Programm möglichst einfach zu gestalten. Aus diesem Grund sind im zweiten Kapitel Worte zusammengestellt, um Daten und Texte in den Programmablauf eingeben zu können. Auch für die Ausgabe von Daten auf den Bildschirm und auf den Drucker werden speziell definierte Worte benötigt.

Das dritte Kapitel zeigt eine Möglichkeit, Zeichenketten in FORTH zu definieren. Hier sind natürlich Worte zum Suchen und Vergleichen von Zeichenketten sehr wichtig. Für die Programmierung von Karteien und Inventarverzeichnissen sind gewisse Datenstrukturen notwendig. Im vierten Kapitel sind deshalb Worte zum Aufbau von verketteten Listen und zum Aufbau von binären Bäumen beschrieben.

Das nächste Kapitel wurde von Herrn Dr. Schmitter bearbeitet und zeigt ein Beispiel für künstliche Intelligenz. Hier werden Entscheidungsbäume aufgestellt und Worte zu Associationen zusammengefasst.

Das Fehlersuchen in Programmen ist oft mühevoll und lästig. Einige Erleichterungen sind in Kapitel sechs angegeben. Hier wird gezeigt, wie ein Wort im Wörterbuch gesucht werden kann und wie ein definiertes Wort dekompiliert wird.

Es folgen eine Sinustabelle und die Angabe von Verfahren zur Programmierung von Rekursionen.

Im nächsten Kapitel ist eine vollständig programmierte Anwendung angegeben. Unter Verwendung einer Adressverwaltung kann leicht ein Programm zur Abonnementverwaltung realisiert werden.

Diese Beispiele werden noch ergänzt durch ein kleines Spielprogramm.

Weiter soll noch das Arbeiten mit dem 6502 Assembler von W. F. Ragsdale gezeigt werden.
Es wird eine Rechnerkopplung mit RS232 und Interruptsteuerung und die Steuerung eines Analog-Digitalwandlers beschrieben.

Zum Schluß ist noch ein vollständiges Programmpaket BUSIPACK angegeben. Hier wird eine Adressverwaltung, kombiniert mit einer Lagerverwaltung und einem Programmteil zum Schreiben von Rechnungen veröffentlicht.

Die in diesem Buch verwendete FORTH-Version entspricht der Fig-FORTH Version 1.1. Der Umfang des Wörterbuches entspricht den Worten, wie sie in der FORTH Encyclopedia von Mitch Derick und Linda Baker beschrieben sind. Dazu kommen noch Worte zur Verarbeitung doppelt langer Zahlen. Diese sind in der folgenden Abbildung angegeben.

Es ist versucht worden, die verwendeten Programme möglichst rechnerunabhängig zu machen. Das ist vor allem dann nicht mehr möglich, wenn es sich um eine formatierte Ausgabe auf den Bildschirm handelt. Im Anhang sind für einige Rechner die Worte für die Cursorsteuerung, Bildschirm löschen und andere maschinenspezifische Befehle angegeben.

```
 0 ( 2WORDS                             EF)
 1 : 2@ ( A-D) DUP 2 + @ SWAP @ ;
 2 : 2! ( DA) SWAP OVER ! 2 + ! ;
 3 : 2+! ( DA) DUP >R 2@ D+ R> 2! ;
 4
 5 : 2DROP DROP DROP ;
 6 : 2SWAP ROT >R ROT R> ;
 7 : 2DUP OVER OVER ;
 8 : 2OVER >R >R 2DUP R> R> 2SWAP ;
 9
10
11
12
13
14   ;S
15
```

1.1 Worte für doppelt lange Zahlen

Notizen

2 Ein- und Ausgabe von Zahlen und Texten

Jedes Programm benötigt Daten, die in den Rechner eingegeben werden und gibt selbst Daten auf den Bildschirm oder auf ein anderes Peripheriegerät aus. In FORTH kann man für diesen Datentransfer den Speicherbereich PAD verwenden. Aber auch andere Lösungen sind denkbar.

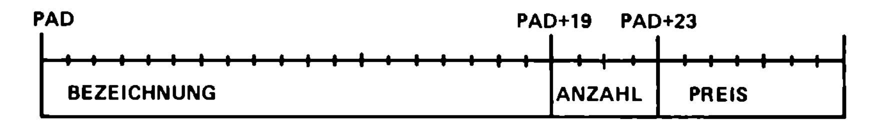
2.1 Eingabe von Daten

Ein kleines Warenverzeichnis hat folgenden Aufbau:

Bezeichnung	19 Stellen
Anzahl	4 Stellen
Preis	7 Stellen

Durch eine passende Eingabe wird ein Datensatz in PAD zusammengesetzt und anschließend auf Diskette gespeichert.

Die Länge des Datensatzes ist 30 Bytes. Durch die Blockstruktur der Datenaufzeichnung auf Diskette muß die Datensatzlänge ein ganzzahliger Teil von 256 oder 1024 sein. Als Länge wird deshalb 32 Bytes gewählt. Die nicht gebrauchten 2 Byte werden in einem späteren Beispiel benötigt. Die folgende Abbildung 2.1 zeigt den Aufbau des Datensatzes in PAD.



2.1 Aufbau eines Datensatzes in PAD

Alle Eingaben werden als Zeichenketten eingegeben. Die Bezeichnung kann beliebiger Text sein. Für die Eingabe der Anzahl sind nur die Ziffern 0 bis 9 erlaubt und bei der Eingabe kann noch ein Dezimalpunkt mit eingegeben werden.

Für die Eingabe werden folgende Worte vereinbart:

```
: S-EIN ( A) 13 WORD HERE COUNT
  ROT PAD + SWAP CMOVE ;

  0 S-EIN KERNSEIFE  OK
  PAD 10 TYPE KERNSEIFE  OK
```

Das Wort S-EIN erwartet auf dem Stapel die relative Adresse zu PAD, ab welcher die folgende Zeichenkette gespeichert wird. Diese Zeichenkette wird nur durch ein Leerzeichen getrennt direkt hinter dem Wort S-EIN eingegeben. Diese Eingabe hat aber den Nachteil, daß die Länge der Eingabe nur durch das RETURN-Zeichen begrenzt ist. Da das nächste Wort nach PAD+19 geschrieben wird, werden Teile der Bezeichnung, wenn sie zu lang sind, überschrieben.

Dies wird bei dem folgenden Wort vermieden.

```
: IP ( NA) PAD + SWAP EXPECT ;

  5 10 IP HEUTE OK
  PAD 10 + 5 TYPE HEUTE OK
```

Das Wort IP verlangt zwei Angaben, die Länge der Eingabe N und die relative Adresse zu PAD, auf dem Stapel. Die Eingabe wird nach N Zeichen oder nach RETURN abgebrochen. Der Text wird ab PAD+A gespeichert.

Eine dritte Art der Eingabe ist die Verwendung des Wortes KEY. Dabei kann jedes eingegebene Zeichen geprüft werden. Diese Eingabe erfolgt nun nicht nach PAD, sondern an einen anderen festen Platz SPAD. Die Adresse von SPAD ist PAD+RECLEN, wobei RECLEN die Länge des Datensatzes ist.

Das Wort ?Z überprüft, ob das eingegebene Zeichen größer 47 und

kleiner 58 ist. Nur dann ist es eine Ziffer. Anstatt mit IF . . . ELSE . . . THEN zu verzweigen, werden mit AND die beiden Abfrageergebnisse zusammengefasst.

```
32 CONSTANT RECLEN
 1 VARIABLE CNT
: SPAD ( -A) PAD RECLEN + ;
: ?Z ( N-NF) DUP 47 > OVER 58 <
  AND ;
: N>P 0 SPAD C! 1 CNT ! 32 SPAD
  C! BEGIN KEY DUP 13 = NOT
     WHILE ?Z IF DUP EMIT SPAD
     CNT @ + C! 1 CNT +!
     ELSE DROP THEN REPEAT
  CNT @ SPAD C! 32 CNT @ SPAD
  + C! DROP ;
```

Das Wort N>P übernimmt eine Zeichenkette vom Tastenfeld und speichert sie ab SPAD+1 ab. Alle Zeichen, die keine gültigen Ziffern darstellen, werden unterdrückt. In das erste Byte von SPAD wird die um Eins erhöhte Länge der Zeichenkette gespeichert. Das letzte Zeichen ist ein Leerzeichen. Damit ist diese Zeichenkette für die Ausgabe mit COUNT TYPE vorbereitet.

```
N>P 12345 OK
SPAD COUNT TYPE 12345  OK
```

Gleichzeitig kann mit NUMBER diese Zeichenkette in eine doppelt lange Zahl gewandelt werden. Das Wort NUMBER erwartet auf dem Stapel die Anfangsadresse der zu wandelnden Zeichenkette und übergibt eine doppelt lange Zahl. Die Zeichenkette muß mit einem Leerzeichen abgeschlossen sein. Eine Fehlermeldung wird ausgegeben, wenn in der Zeichenkette ein Zeichen enthalten ist, das keine Ziffer darstellt. Davon ausgenommen ist das Minuszeichen und der Dezimalpunkt.

```
SPAD NUMBER  OK
.S
12345 0  OK
```

Das Wort P>P ist genauso definiert wie das Wort N>P, nur wird hier noch der Dezimalpunkt mit zugelassen.

```
: P>P 0 SPAD C! 1 CNT ! 32 SPAD
  C! BEGIN KEY DUP 13 = NOT
     WHILE ?Z OVER 46 = OR
     IF DUP EMIT SPAD
     CNT @ + C! 1 CNT +!
     ELSE DROP THEN REPEAT
  CNT @ SPAD C! 32 CNT @ SPAD
  + C! DROP ;
```

Nun können für die Eingabe in das Warenverzeichnis die folgenden Worte vereinbart werden:

```
: SCLR SPAD RECLEN 32 FILL ;
: PCLR  PAD RECLEN 32 FILL ;
: BEZ> 19 0 IP ;
: ZA> SPAD N>P SPAD 1+ PAD 19 +
  4 CMOVE ;
: PR> SPAD P>P SPAD 1+ PAD 23 +
  7 CMOVE ;   ;S
```

Das Wort BEZ> speichert eine 19-stellige Bezeichnung ab Adresse PAD. ZA> speichert eine Zeichenkette, die nur Ziffern als gültige Zeichen ab Adresse PAD+19 enthält und das Wort PR> speichert eine Ziffernfolge mit Dezimalpunkt ab Adresse PAD+23.

```
SCLR  OK
PCLR  OK
BEZ> SONNENBLUMEN  OK
ZA> 100 OK
PR> 2.50 OK
PAD 32 TYPE SONNENBLUMEN    100 2.50      OK
```

In den beiden Worten N>P und Z>P gibt es bei der Eingabe keine Korrekturmöglichkeit. Diese muß für jeden Rechner gesondert programmiert werden. Das liegt daran, daß bei den verschiedenen Home-

computern unterschiedlicher Code für das BACKSPACE-Zeichen verwendet wird. Ebenso wird die Cursorsteuerung unterschiedlich behandelt. Ein Beispiel für eine solche Programmierung zeigt das folgende Textfeld. Hier wurde die Korrektur mit der Backspacetaste durchgeführt und der Cursor mit −1 211 C! eine Stelle zurückgesetzt.

```
SCR # 36
  0  ( INPUT WORDS                    ef)
  1  0 VARIABLE CNT
  2  : PADC PAD 128    32 FILL ;
  3 : ?Z ( N-NF) DUP 47 > OVER 58 <
  4   AND OVER 46 = OR ;
  5 : (S) 32 C, -1 ALLOT ;
  6 : (Z> ( -D) CNT @ HERE OVER -
  7   SWAP 0= NOT IF NUMBER ELSE
  8   DROP THEN ;
  9 : (Z>) ( -D) 1 CNT ! (S) BEGIN
 10   KEY DUP 13 = NOT WHILE ?Z IF
 11   DUP EMIT C, (S) 1 CNT +! ELSE
 12  20 = IF -1 ALLOT -1 CNT +! (S)
 13 -1 211 +! THEN THEN REPEAT DROP
 14   (Z> CNT @ 1 - MINUS DP +! ;
 15
 OK
```

Eingabe von Zahlen mit Korrekturmöglichkeit beim C-64

2.2 Ausgabe von Daten

Bei einigen Heimcomputern wird bei der Ausgabe des ASCII-Zeichens 00 auf dem Bildschirm ein Grafiksymbol ausgegeben. Dies führt dann zu einem unschönen Bildschirmausdruck, wenn Text mit EXPECT eingegeben und mit TYPE wieder ausgedruckt wird. Durch das Wort ATYPE kann dies umgangen werden.

```
: ATYPE -DUP IF OVER + SWAP
  DO I C@ 127 AND DUP 0=
  IF DROP ELSE EMIT THEN
  LOOP ELSE DROP ENDIF ;
```

Für die Formatierte Ausgabe auf dem Bildschirm oder einem Drucker werden die folgenden Worte definiert:

```
0 VARIABLE H 0 VARIABLE V

: ATYPE -DUP IF OVER + SWAP
  DO I C@ 127 AND DUP 0=
  IF DROP ELSE EMIT 1 H +! THEN
  LOOP ELSE DROP ENDIF ;
```

Es werden zwei Variable H und V vereinbart. In der Variablen H ist die horizontale Position und in V die vertikale Cursorposition oder die Position des Druckkopfes eines Druckers gespeichert. Die Variable H wird bei der Ausgabe eines Zeichens in ATYPE jeweils um Eins erhöht. Falls dieses Wort nicht benötigt wird, kann TYPE durch NTYPE ersetzt werden.

```
: NTYPE DUP H +! TYPE ;
: HO ( N) 0 DO SPACE 1 H +!
  LOOP ;
: VE ( N) 0 DO CR 1 V +! LOOP
  0 H ! ;
: ADJ ( N) H @ - DUP 0 > IF HO
  THEN ;
: RADJ ( ANN'N") SWAP ADJ SWAP
  DUP >R - HO R> ATYPE ;
```

Das Wort HO gibt N Leerzeichen aus, während das Wort VE N Zeilenvorschübe macht. Dabei wird gleichzeitig H auf Null gesetzt. Das Wort ADJ setzt den Druckkopf auf die horizontale Stelle N.

Das Wort RADJ druckt einen Text, der ab der Adresse A mit der Länge N gespeichert ist, rechtsbündig in einer Spalte der Breite N'' aus. Der Beginn der Spalte ist N'.

```
0 H !  OK
CR  PAD 10 EXPECT
123.45  OK
PAD 6 10 20 RADJ                          123.45 OK
0 H !  OK
PAD 10 EXPECT 1.23  OK
PAD 4 10 20 RADJ                            1.23 OK
```

Das Wort TEX> druckt einen Text, der ab Adresse A mit der Länge N gespeichert ist, linksbündig ab der Stelle N' aus. Mit diesen Worten erhält man dann folgendes Druckbild:

```
: TEX> ( ANN') ADJ -TRAILING
  ATYPE ;

  PAD 9 EXPECT GUTEN TAG OK
  0 H !  OK
  PAD 9 5 TEX>      GUTEN TAG OK
  0 H !  OK
  PAD 9 8 TEX>         GUTEN TAG OK
```

In den meisten Fällen werden die zu druckenden Zahlen Ergebnisse einer Berechnung sein. Vor einem formatierten Ausdruck müssen diese Zahlen in Zeichenketten gewandelt werden.

Die folgenden Worte zeigen einige Beispiele:

```
: #N ( d)    <# #S #> ;
: #DM ( d) <# # # 46 HOLD #S
  #> ;
: #% ( d) <# 37 HOLD # 46 HOLD
  #S #> ;
: #DA ( d) <# # # 46 HOLD # #
  46 HOLD #S #> ;
```

Einfach lange Zahlen müssen vor der Wandlung in eine doppelt lange Zahl gewandelt werden. Falls es sich um positive Zahlen handelt, wird einfach eine Null auf dem Stapel abgelegt. Sonst kann auch mit S->D gewandelt werden. Nach der Umwandlung in eine Zeichenkette ist die Anfangsadresse und die Länge dieser Zeichenkette auf dem Stapel.

Mit RADJ kann diese Zahl passend im Ausdruck plaziert werden.

```
N>P 1234 OK
2 VE SPAD NUMBER #DM 5 15 RADJ

               12.34 OK
N>P 12345 OK
2 VE SPAD NUMBER #DM 5 15 RADJ

              123.45 OK
N>P 280384 OK
2 VE SPAD NUMBER #DA 5 TEX>

     28.03.84 OK
N>P 140 OK
2 VE SPAD NUMBER #% 5 TEX>

     14.0% OK
N>P 1234567 OK
2 VE SPAD NUMBER #N 5 15 RADJ

             1234567 OK
N>P 34 OK
2 VE SPAD NUMBER #N 5 15 RADJ

                  34 OK
```

Sollen auf ein Formular Texte gedruckt werden, die immer gleich sind, so kann man diese Texte im Programm speichern. Muß später ein Text geändert werden, so ist das Programm neu zu kompilieren. Einfacher ist es, diese Texte auf Diskette zu speichern. Sie werden in einen nicht benötigten Block geschrieben und später von dort geholt und ausgedruckt.

```
: S>D ( NA) BLOCK + 13 WORD HERE
  COUNT >R SWAP DUP R SWAP C!
  1+ R> CMOVE UPDATE ;

: S<D ( NA) BLOCK + COUNT TYPE ;

: DTEX> ( NAN') ADJ BLOCK +
  COUNT -TRAILING ATYPE ;
```

Das Wort S>D erwartet zwei Angaben auf dem Stapel. A ist die Nummer des Blockes, in welchen der Text geschrieben werden soll, und N ist die zum Blockanfang relative Adresse, ab welcher der Text gespeichert werden soll. In dem ersten Byte wird die Länge des Textes gespeichert.

Das Wort S<D holt den Text von der Diskette. Die Zahlen N und A haben die gleiche Bedeutung wie beim Abspeichern.

Mit DETX> wird der Text von der Diskette gelesen und linksbündig ab der Druckposition N' ausgegeben.

```
0 36 S>D DIESER TEXT WIRD GESPEICHERT.  OK
0 36 S<D DIESER TEXT WIRD GESPEICHERT. OK
7 36 S<D TEXT WIRD GESPEICHERT. OK
1 VE 0 36 5 DTEX>
     DIESER TEXT WIRD GESPEICHERT. OK
1 VE 0 36 10 DTEX>
          DIESER TEXT WIRD GESPEICHERT. OK
```

Beim Eingeben von Text in die Textfelder können die folgenden Worte verwendet werden.

Dabei werden die im Kernel definierten Worte (LINE) und .LINE verwendet. Die Definition von (LINE) ist:

(LINE) (NN'–AN'')

(LINE) übernimmt vom Stapel die Zeilennummer N und die Numer N' des Textfeldes. Die Anfangsadresse der Zeile im Speicher und die Länge der Zeile bleiben auf dem Stapel.

SCR ist eine Systemvariable, welche die Nummer des augenblicklichen Textfeldes enthält. Im Wort LIST wird die Nummer des Textfeldes in SCR gespeichert.

```
26 LIST
 SCR # 26
   0 ( TEXT AUF DISK  CNTD    1.3 EF)
   1
   2 : P ( N) SCR @ (LINE) OVER SWAP
   3   BLANKS 0 WORD HERE COUNT 32
   4   MIN ROT SWAP CMOVE UPDATE ;
   5
   6
   7
   8
7 P TEXT NACH ZEILE 7  OK
26 LIST
SCR # 26
  0 ( TEXT AUF DISK  CNTD    1.3 EF)
  1
  2 : P ( N) SCR @ (LINE) OVER SWAP
  3   BLANKS 0 WORD HERE COUNT 32
  4   MIN ROT SWAP CMOVE UPDATE ;
  5
  6
  7 TEXT NACH ZEILE 7
  8


  2 26 .LINE : P ( N) SCR @ (LINE) OVER SWAP OK
```

Nachtrag:
Das auf Seite 6 angegebene Wort IP kann unter Umständen schon vorhandene Einträge in PAD überschreiben, da EXPECT Nullen an die Eingabe anhängt. Die Neudefinition von IP vermeidet dies.

```
( IP NEU                      EF)
: PADD PAD 128 + ;
: PADDC PADD 128 32 FILL ;
: IP ( NA) PADDC OVER PADD SWAP
  EXPECT PAD + SWAP PADD ROT
  ROT CMOVE ;
```

3 Zeichenketten in FORTH

In FORTH sind keine Worte für die Verarbeitung von Zeichenketten vorhanden. Diese können aber leicht definiert werden. Ausgehend von den Worten ." und (.") werden die Worte " und ["] definiert.

```
HEX
: ["] ( -AL) R COUNT DUP 1+ R>
  + >R ;

: " ( -AL) 22 STATE @ IF COMPILE
  ["] WORD HERE C@ 1+ ALLOT
  ELSE WORD HERE DUP C@ 1+ PAD
  SWAP CMOVE PAD COUNT THEN ;
  IMMEDIATE
```

Das Wort " eröffnet eine Zeichenkette. Diese endet wiederum mit einem ". Das Wort hat unterschiedliches Compile- und Laufzeitverhalten. Wird dieses Wort innerhalb einer Doppelpunkt Definition verwendet, so wird der Text im Parameter-Feld des definierten Wortes gespeichert. Während der Laufzeit wird der Text aus dem Wörterbuch gelesen und nach PAD übertragen. Gleichzeitig wird auf dem Stapel die Adresse von PAD und die Länge des Textes abgelegt. Die direkte Eingabe von

" DAS IST TEXT"

speichert diese Zeile ab der Adresse PAD.

```
" DAS IST TEXT"  OK

PAD COUNT TYPE DAS IST TEXT OK
```

Für die Speicherung von Zeichenketten wird das Definitionswort STRING verwendet. Während des compilierens wird das Längenbyte gespeichert und für N Bytes Speicherplatz im Wörterbuch reserviert. Während der Laufzeit wird die Adresse des Längenbytes und die Länge selbst auf dem Stapel abgelegt.

```
: STRING ( N) <BUILDS 1 MAX
   FF MIN DUP C, 0 C, ALLOT
   DOES> 1+ COUNT ;
```

Die Eingabe von

```
20 STRING A$
```

eröffnet eine Variable A$ für eine Zeichenkette mit maximal 20 Zeichen. In diese Variable kann nun Text mit

```
" ABCDEFGHUJ" A$ S!
```

gespeichert werden.

Das Wort S! speichert Text. Die Definition lautet:

S! (ana'n')

Der Text ab Adresse a mit der Länge n wird ab Adresse a' in einer Zeichenkettenvariablen gespeichert. Dabei werden nur so viele Zeichen eingegeben, für die in der Variablen Platz gelassen wurde.

```
: S! ( ALA'L') DROP DUP 2 - C@
  ROT MIN DUP >R OVER R> SWAP
  1 - C! CMOVE ;
```

```
20 STRING Z$  OK
" GUTEN TAG" Z$ S!  OK
Z$ TYPE GUTEN TAG OK
```

```
20 STRING B$
: >B$ " TEST TEST TEST" B$ S! ;
30 STRING C$
" ABCDEFGHIJKLMNOPQRSTUVWXYZ"
C$ S!
```

Das Wort >B$ speichert dreimal das Wort TEST in der Variablen B$. Die Buchstaben A bis Z werden in der Variablen C$ gespeichert.

```
: S+ ( ANA'N'-A"N") 2SWAP PAD 1+
  SWAP DUP >R CMOVE R> 2DUP +
  PAD C! PAD 1+ + SWAP CMOVE
  PAD COUNT ;
```

```
10 STRING E$  OK
" GUTEN" E$ S!  OK
10 STRING F$  OK
"  MORGEN" F$ S!  OK
E$ F$ S+  OK
PAD COUNT TYPE GUTEN MORGEN OK
```

Das Wort S+ "addiert" zwei Zeichenketten. Vor dem Aufruf dieses Wortes sind die Anfangsadresse und die Länge der ersten und zweiten Zeichenkette auf dem Stapel. Nach der Ausführung des Wortes ist die Anfangsadresse der zusammengesetzten Zeichenkette und deren Länge auf dem Stapel. Beide Zeichenketten werden in PAD zusammengesetzt. In der Adresse PAD wird die Länge, ab PAD 1+ wird die Zeichenkette gespeichert.

```
: LEN ( AN-N) SWAP DROP ;

: LEFT$ ( ANN'-AN') LEN ;

: MID$ ( ANNN"-A'N"') >R >R DROP
  R> + R> ;

: RIGHT$ ( ANN'-A'N") >R + R -
  R> ;
```

Das Wort LEN bestimmt die Länge einer Zeichenkette. Da der Aufruf einer Zeichenketten-Variablen die Adresse und die Länge auf dem Stapel ablegt, braucht zur Bestimmung der Länge nur die Adresse vom Stapel entfernt werden.

```
C$ LEN . 26  OK
```

Das Wort LEFT$ begrenzt die Zeichenkette auf N' Zeichen vom linken Rand aus.

```
C$ 5 LEFT$ TYPE ABCDE OK
```

Das Wort MID$ bestimmt eine Teilkette, die beim Zeichen N beginnt und N" Zeichen lang ist.

```
C$ 3 4 MID$ TYPE DEFG OK
```

Das letzte Wort RIGHT$ begrenzt eine Zeichenkette auf N' Zeichen vom rechten Rand aus.

```
C$ 5 RIGHT$ TYPE VWXYZ OK
```

Beispiel

```
: $T ( NN') CLR DO C$ I LEFT$
  TYPE CR LOOP ;

30 STRING L$
"                          " L$ S!
```

Das Wort $T erzeugt den folgenden Ausdruck:

```
CR 21 1 $T
A
AB
ABC
ABCD
ABCDE
ABCDEF
ABCDEFG
ABCDEFGH
ABCDEFGHI
ABCDEFGHIJ
ABCDEFGHIJK
ABCDEFGHIJKL
ABCDEFGHIJKLM
ABCDEFGHIJKLMN
ABCDEFGHIJKLMNO
ABCDEFGHIJKLMNOP
ABCDEFGHIJKLMNOPQ
ABCDEFGHIJKLMNOPQR
ABCDEFGHIJKLMNOPQRS
ABCDEFGHIJKLMNOPQRST
 OK
```

und das Wort $T2 den Ausdruck

```
: $T2 ( NN') CLR DO L$ 20 I -
  LEFT$ TYPE C$ I RIGHT$ TYPE
  CR LOOP ;
```

```
CR 20 1 $T2
                  Z
                 YZ
                XYZ
               WXYZ
              VWXYZ
             UVWXYZ
            TUVWXYZ
           STUVWXYZ
          RSTUVWXYZ
         QRSTUVWXYZ
        PQRSTUVWXYZ
       OPQRSTUVWXYZ
      NOPQRSTUVWXYZ
     MNOPQRSTUVWXYZ
    LMNOPQRSTUVWXYZ
   KLMNOPQRSTUVWXYZ
  JKLMNOPQRSTUVWXYZ
 IJKLMNOPQRSTUVWXYZ
HIJKLMNOPQRSTUVWXYZ
OK
```

Bei der Definition obiger Worte ist, wie in FORTH üblich, keinerlei Überprüfung der Längen der Zeichenketten programmiert worden. Der Programmierer muß selbst aufpassen, daß die richtigen Parameter übergeben werden. Eine Prüfung der Längen von Zeichenketten kann nachträglich programmiert werden, allerdings wird die Laufzeit der Zeichenketten Bearbeitung dann länger.

Die Worte für den Vergleich von Zeichenketten werden am besten in Assembler geschrieben. Damit wird eine schnelle Ausführungszeit erreicht. Da diese Worte aber für jede CPU anders geschrieben werden müssen, soll hier ein Wort, (VERGL), gezeigt werden, das zwei Zeichenketten auf Gleichheit prüft und in FORTH geschrieben ist.

```
( VERGLEICH VON ZEICHENKETTEN  )
: (VERGL) ( AA'C-F)
  BEGIN ROT DUP C@ >R OVER I =
     R> SWAP DUP IF 0 ELSE DROP
     >R ROT DUP C@ R> = DUP DUP
     THEN WHILE 2DROP 1+ >R 1+
     R> ROT REPEAT >R 2DROP
     2DROP R> ;

: S= ( ANA'N'-F)
  DROP SWAP DROP 32 (VERGL) ;
```

Das Wort (VERGL) vergleicht zwei Zeichenketten, deren Anfangsadressen a und a' sind, bis zum Begrenzer c in der Zeichenkette ab der Adresse a. Sind die Zeichenketten bis zum Begrenzer gleich, so ist f=1, andernfalls Null. Das Wort S= übernimmt vom Stapel die Anfangsadresse und die Länge von zwei Zeichenketten und vergleicht sie bis zum Begrenzer 32 (Leerzeichen).

COMPARE ist ein anderes Wort zum Vergleich von Zeichenketten. Auf dem Stapel sind die beiden Anfangsadressen der Zeichenketten und die Länge N, bis z welcher die beiden Zeichenketten verglichen werden sollen.
Nach dem Vergleich ist das oberste Element des Stapels positiv, Null oder negativ. F ist positiv, wenn die Zeichenkette bei der Adresse A alphabetisch kleiner ist, als die Zeichenkette bei der Adresse A'. Der

Wert von F ist Null, wenn beide Zeichenketten gleich sind, und F ist positiv, wenn die Zeichenkette bei A alphabetisch größer ist.

```
: COMPARE ( AA'N-F)
  OVER + SWAP DO COUNT I C@
  - -DUP IF SWAP 0= LEAVE THEN
  LOOP IF 0 THEN ;

" OTTO" F$ S!  OK

" ADAM" E$ S!  OK

E$ DROP F$ COMPARE . -14  OK
" ZUNDER" E$ S!  OK

E$ DROP F$ COMPARE . 11  OK

" OTTO" E$ S!  OK



E$ DROP F$ COMPARE . 0  OK
```

Die Programmierung dieses Wortes ist sehr trickreich und ist (3), VO15, N3, S12 entnommen. Besonders fällt die Verwendung des Wortes COUNT auf. Nach dem Beginn der DO Schleife ist nur noch die Adresse der ersten Zeichenkette auf dem Stapel. Durch COUNT wird diese Adresse um Eins erhöht und das Byte, das bei der alten Adresse gespeichert ist, wird auf dem Stapel abgelegt.

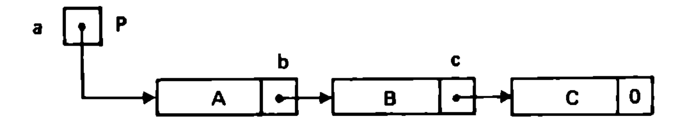
Notizen

4 Datenstrukturen

In diesem Kapitel sollen zwei Datenstrukturen betrachtet werden. Dies sind einmal verkettete Listen und zum anderen binäre Bäume. Solche Datenstrukturen lassen sich in FORTH sehr leicht programmieren, da der Aufbau des Datensatzes vollkommen frei wählbar ist. Gewisse Einschränkungen, die durch das blockweise Aufzeichnen von Daten auf Diskette gegeben sind, können leicht umgangen werden.

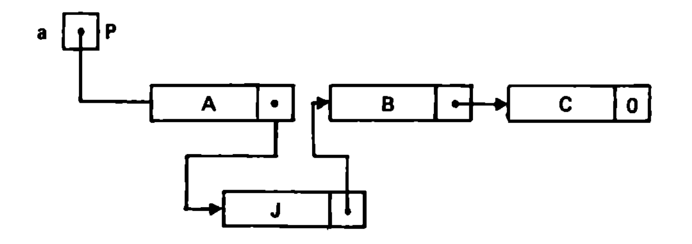
4.1 Verkettete Listen

Die Abbildung 4.1 zeigt eine verkettete Liste. Ein Datensatz enthält ein Schlüsselwort KW und einen Zeiger P.



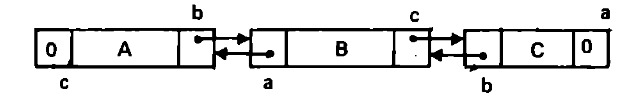
4.1 Verkettete Liste

Bei einer sortierten Liste verweist der Zeiger immer auf das nächst größere Element. Ein Anfangszeiger PA zeigt auf das kleinste Element der Liste. Soll ein neues Element eingefügt werden, so wird durch Vergleichen der Schlüsselworte mit dem neuen Eintrag der Platz für diesen gesucht und durch Einsetzen der Zeiger dieser in die Liste eingetragen. Damit ist er logisch in die Liste eingefügt, physikalisch kann er irgendwo im Speicher abgelegt sein. Das Einfügen zeigt Abbildung 4.2.



4.2 Einfügen eines Elementes

Auf die gleiche Weise kann er aus der Liste entfernt und damit gelöscht werden. Diese einfach verkettete Liste hat den Nachteil, das sie immer nur vom kleinsten Element ausgehend, in einer Richtung durchlaufen werden kann. Für ein praktisches Beispiel soll eine doppelt verkettete Liste benutzt werden. Diese verfügt über zwei Zeiger. Der eine Zeiger, PL, zeigt immer auf den Vorgänger, das nächst kleinere Element, der andere, PR, immer auf den Nachfolger, das nächst größere Element. Eine solche Liste zeigt Abbildung 4.3.



4.3 Doppelt verkettete Liste

Die Abbildung 4.3a zeigt den Aufbau eines Elementes, wie er für das Beispiel einer Kartei mit einem Schlüsselwort verwendet werden soll.



4.3a Aufbau des Schlüsselwortes

Die Länge des Elementes sind 32 Bytes. Davon sind 26 Bytes für das Schlüsselwort frei. Die restlichen sechs Bytes sind drei Zeiger. Dies sind die beiden Zeiger PL und PR, die für die Verkettung benötigt werden

und der Zeiger N. Dieser zeigt auf den Speicherplatz, in welchem die zu dem Schlüsselwort zugehörige Information gespeichert ist.

Beispiele für eine solche Kartei sind Lagerverwaltungen, Personal- oder Patienten Dateien. Bei einer Lagerverwaltung ist die Lagernummer das Schlüsselwort. In einem Textfeld ist die zugehörige Information, der Lagerbestand, Ein- und Verkaufspreis usw. gespeichert. Bei einer Patientenkartei wird der Name als Schlüsselwort verwendet. In dem folgenden Programm soll hauptsächlich das Aufstellen einer Liste gezeigt werden. Die zugehörige Information wird als Text in zugeordnete Textfelder mit dem Editor des FORTH Systems eingetragen.

Den Aufbau der Kartei auf der Diskette zeigt Abbildung 4.4.

Die Schlüsselworte sind ab Block 1 gespeichert. Bei dem verwendeten FORTH ist die Zahl B/BUF gleich 256. Ab Textfeld 40 wird der Text gespeichert.

Die ersten 32 Bytes in Block 1 enthalten in den ersten beiden Bytes die Zahl N. Dies ist die laufende Nummer für den nächsten Eintrag. Die nächsten beiden Bytes enthalten den Zeiger PA, der auf das kleinste Element in der Liste zeigt. Eine Liste ist leer, wenn N gleich Eins und PA gleich Null ist.

Ein neuer Eintrag wird durch

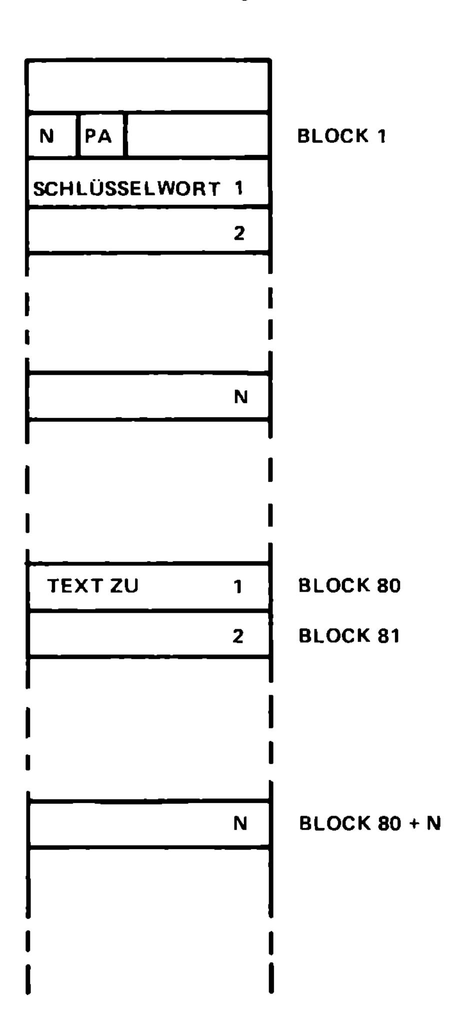
NEW ⟨SCHLUESSELWORT⟩

in die Liste aufgenommen. Nach dem Verketten wird über dem Editor das zugehörige Textfeld auf dem Bildschirm angezeigt und zur Eingabe von Text vorbereitet.

Das Programm beginnt in Textfeld 40. Das Wort #INDEX bestimmt die Speicheradresse für den N-ten Eintrag. Die Länge eines Elementes dieser Liste ist 32 Bytes.
Die Länge eines Blockes ist aber ein ganzzahliges Vielfaches dieser Zahl.

Die Worte mit dem ' Zeichen holen Zeiger und Zahlen, Worte mit dem

! Zeichen speichern diese an die richtige Stelle zurück.



4.4 Aufbau der Kartei auf Diskette

Für den Vergleich der Schlüsselworte wird COMPARE verwendet. Das Vergleichswort ist in PAD, das Schlüsselwort wird aus der Liste geholt. Das Wort !MEM speichert ein Element, nachdem es verkettet wurde, an das Ende der Liste. EMPTY-LIST erzeugt eine leere Liste.

Beim Eintragen eines Elementes müssen folgende Fälle betrachtet

werden:

1. Die Liste ist leer. PA=0, N=1.

2. Ein Element ist in der Liste, dann ist PA=1, N=2, PL(1)=0 und PR(1)=0.

3. Der neue Eintrag ist größer, als alle anderen Elemente. J ist die Nummer des neuen Eintrags und K die Nummer des Vorgängers. Dann ist PR(J)=0, PL(J)=K und PR(K)=J.

4. Der neue Eintrag ist kleiner als alle anderen Elemente. K ist die Nummer des Nachfolgers. Dann ist PL(J)=0, PR(J)=K, PL(K)=J und PA=J.

5. Der neue Eintrag muß zwischen dem Element K und dem Element J eingefügt werden. Dann wird PR(K)=J, PL(J)=K, PR(J)=I und PL(I)=J.

Das Wort ?F# stellt fest, ob die Liste leer ist, und fügt das erste Element ein. Die weitere Verkettung übernimmt das Wort (⟩LST). Dieses ist der besseren Übersicht nochmals getrennt in Abbildung 4.5 gezeigt.

Durch

SUCHE ⟨SCHLUESSELWORT⟩

kann ein Eintrag auf dem Bildschirm angezeigt werden. Am oberen Bildschirmrand wird das Schlüsselwort, darunter der zu diesem eingegebene Text angezeigt. Von dieser Stelle aus kann nun mit der Eingabe von N ⟨RET⟩ zum Nachfolger und mit V ⟨RET⟩ zum Vorgänger geblättert werden. Das linke Ende der Liste ist mit PL=0 und das rechte Ende mit PR=0 angezeigt. Werden diese beim Blättern erreicht, so wird "ENDE DER LISTE" ausgegeben.

Ein Eintrag kann mit

LOESCHE ⟨SCHLUESSELWORT⟩

aus der Kartei gelöscht werden. Die dabei entstehende Lücke kann mit dem letzten Element aufgefüllt werden, wobei die Zeiger neu gesetzt werden.

```
: (>LST) @PA
  BEGIN @PTRS COMP DUP 0=
   IF ." SCHON IN LISTE" DROP 0 1
   ELSE 0<
    IF PJR @ 0=
     IF ( NEUES EL IST GROESSTES )
      0 !NPR @N PJ @ !PR PJ @ !NPL 1 1
     ELSE ( WEITERSUCHEN ) PJR @ 0
     THEN
    ELSE PJL @ 0=
     IF ( NEUES EL IST KLEINSTES )
      0 !NPL @PA !NPR @N DUP @PA !PL !PA 1 1
     ELSE ( EINSORTIEREN )
      PJ @ @PL !NPL PJL @ @PR !NPR
      @N PJL @ !PR @N PJ @ !PL 1 1
     THEN
    THEN
   THEN
  UNTIL IF !MEM THEN ;
```

4.5 Das Wort (>LST)

```
40 48 >P 0 PR#
SCR # 40
  0 ( VKL                     10.3.84 EF)
  1 1 CONSTANT START
  2 32 CONSTANT RECLEN
  3 0 VARIABLE PJL 0 VARIABLE PJR
  4 0 VARIABLE PJ
  5 : @PA ( -N) START BLOCK 2 + @ ;
  6 : !PA ( N) START BLOCK 2 + !
  7   UPDATE ;
  8 : #INDEX ( N-A) RECLEN B/BUF
  9  */MOD START + BLOCK + ;
 10 : @PL ( N-N') #INDEX 26 + @ ;
 11 : @PR ( N-N') #INDEX 28 + @ ;
 12 : @N ( -N) 0 #INDEX @ ;
 13 -->
 14
 15
```

```
SCR # 41
  0 ( VKL CNTD              10.3.84 EF)
  1 : !NPL ( N) PAD 26 + ! ;
  2 : INPR ( N) PAD 28 + ! ;
  3 : !PL ( NN') #INDEX 26 + !
  4   UPDATE ;
  5 : !PR ( NN') #INDEX 28 + !
  6   UPDATE ;
  7 : !N ( N) 0 #INDEX ! UPDATE ;
  8 : +N 1 0 #INDEX +! UPDATE ;
  9 : @NR ( N-N') #INDEX 30 + @ ;
 10 : !NR @N PAD 30 + ! ;
 11
 12 -->
 13
 14
 15
```

```
SCR # 42
  0 ( VKL COMPARE           10.3.84 EF)
  1 : COMPARE ( AA'N-F) ( F -0+ )
  2   OVER + SWAP
  3   DO COUNT I C@ - -DUP
  4    IF SWAP 0= LEAVE THEN
  5   LOOP IF 0 THEN ;
  6
  7 : COMP ( N-F) #INDEX PAD 26
  8   COMPARE ;
  9
 10 -->
 11
 12
 13
 14
 15
```

```
SCR # 43
  0 ( VKL CNTD              10.3.84 EF)
  1 : !MEM !NR PAD @N #INDEX
  2   RECLEN CMOVE UPDATE +N ;
  3
  4 : EMPTY-LIST 0 !PA 1 !N ;
  5 : ?F# ( -F) 0 @PA 0= IF !MEM 1+
  6   DUP !PA 0 1 !PL 0 1 !PR THEN ;
  7
  8 : @PTRS ( N-N) DUP 2DUP @PL
  9   PJL ! @PR PJR ! PJ ! ;
 10
 11 : @MEM ( N) #INDEX PAD RECLEN
 12   CMOVE ;
 13
 14 -->
 15
```

```
SCR # 44
  0 ( VKL >LST              10.3.84 EF)
  1 : (>LST) @PA
  2   BEGIN @PTRS COMP DUP 0= IF
  3   ." SCHON IN LISTE" DROP 0 1
  4   ELSE 0<  IF PJR @ 0= IF
  5 ( NEUES EL IST GROESSTES) 0 !NPR
  6   @N PJ @ !PR PJ @ !NPL 1 1
  7   ELSE ( WEITERSUCHEN) PJR @ 0
  8   THEN ELSE PJL @ 0= IF
  9 ( NEUES EL IST KLEINSTES) 0 !NPL
 10  @PA !NPR @N DUP @PA !PL !PA 1 1
 11   ELSE ( EINSORTIEREN)
 12   PJ @ @PL !NPL PJL @ @PR !NPR
 13  @N PJL @ !PR @N PJ @ !PL
 14  1 1 THEN THEN THEN UNTIL IF
 15   !MEM THEN ;  -->
```

```
SCR # 45
  0 ( VKL CNTD                10.3.84 EF)
  1 : PADC PAD RECLEN 32 FILL ;
  2 : >PAD PADC 13 WORD HERE COUNT
  3   PAD SWAP CMOVE ;
  4 : NEW >PAD ?F# NOT IF (>LST)
  5   THEN ;
  6
  7 : .REC ( N) CR @MEM PAD 19 TYPE
  8   CR PAD 26 + @ . PAD 28 + @ .
  9   PAD 30 + @ . CR ;
 10
 11 : PRINT @PA BEGIN DUP 0= NOT IF
 12   DUP .REC @PR DUP THEN 0=
 13   UNTIL DROP ;
 14 : .MES2 ." ENDE DER LISTE" CR ;
 15 -->
```

```
SCR # 46
  0 ( VKL CNTD                30.3.84EF)
  1 : (J/N) ." ( /N) " KEY  78 =
  2   IF 0 ELSE 1 THEN ;
  3 : <-> ( N) ." V N X "
  4   BEGIN KEY DUP 86 =
  5    IF DROP DUP @PL 0= NOT
  6     IF @PL ELSE .MES2 THEN 1
  7    ELSE DUP 78 =
  8     IF DROP DUP @PR 0= NOT
  9      IF @PR ELSE .MES2 THEN 1
 10     ELSE 88 =
 11      IF DROP 0 ELSE 1 THEN
 12     THEN
 13    THEN
 14   WHILE DUP .REC REPEAT ;
 15   -->
```

```
SCR # 47
  0 ( VKL CNTD SUCHEN     30.3.84EF)
  1 : .MES1 ." NICHT IN LISTE" ;
  2 : (SUCHE) ( -N) @PA
  3    BEGIN DUP COMP 0=
  4     IF DUP .REC 1
  5     ELSE DUP @PR 0=
  6      IF .MES1 DROP 1
  7      ELSE @PR 0
  8      THEN
  9     THEN
 10    UNTIL ;
 11
 12 : SUCHE >PAD (SUCHE) ." BLAETTER
 13 N" (J/N) IF <-> ELSE DROP THEN ;
 14
 15 -->
```

```
SCR # 48
  0 ( VKL CNTD LOESCHEN 30.3.84  EF)
  1 : (LOESCH) ( N)
  2 DUP DUP @PR 0=
  3 IF @PL 0 SWAP !PR
  4 ELSE DUP @PL 0=
  5   IF @PR DUP 0 SWAP !PL !PA
  6   ELSE DUP @PL OVER @PR !PL
  7     DUP @PR SWAP @PL !PR
  8   THEN
  9 THEN DUP 0 SWAP !PR 0 SWAP !PL ;
 10
 11 : LOESCHE >PAD (SUCHE) (J/N)
 12   IF (LOESCH) ELSE DROP THEN ;
 13
 14
 15 ;S
```

4.6 Verkettete Listen

Ein Beispiel:

```
       EMPTY-LIST  OK                     NEW BERND  OK
         OK                               PRINT
       NEW OTTO  OK                       ADAM
       NEW KARL  OK                       0 6 4
       NEW HEINZ  OK
       NEW ADAM  OK                       BERND
       NEW ZAUSEL  OK                     4 3 6
       PRINT
       ADAM                               HEINZ
       0 3 4 → N                          6 2 3
  PL ──┘ └── PR
       HEINZ                              KARL
       4 2 3                              3 1 2

       KARL                               OTTO
       3 1 2                              2 5 1

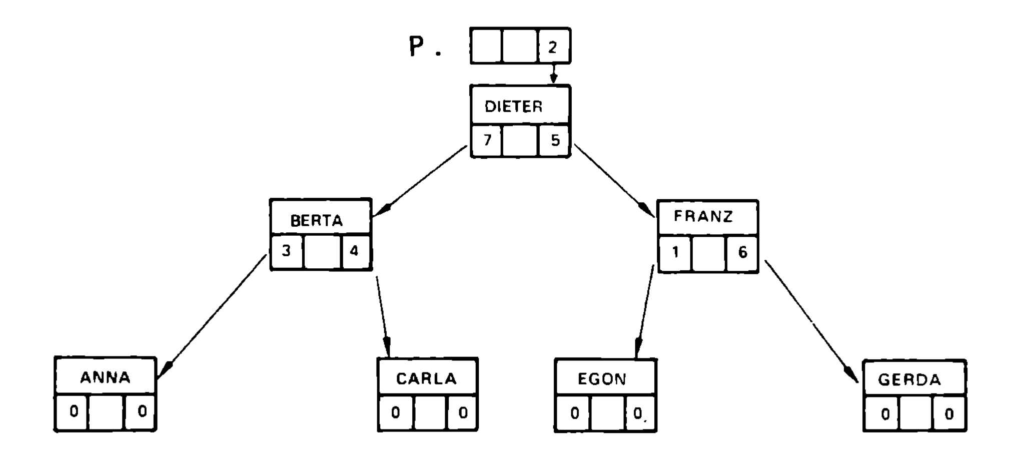
       OTTO                               ZAUSEL
       2 5 1                              1 0 5
                                           OK
       ZAUSEL
       1 0 5
        OK
```

Eine andere Verwendung dieses Programms ist die Verkettung von Namen aus der Adressverwaltung im nächsten Kapitel.

Diese verketteten Listen sind einfach zu programmieren, haben jedoch einen entscheidenden Nachteil. Bei der Suche nach einem Eintrag wird beim kleinsten Element begonnen und dann linear durch die Liste gelaufen, bis der entsprechende Eintrag gefunden ist. Bei sehr langen Listen stellt dies einen erheblichen Zeitaufwand dar. Soll in einer großen Datenmenge schnell auf einen Eintrag zugegriffen werden, so wird als Datenstruktur ein binärer Baum verwendet.

4.2 Binäre Bäume

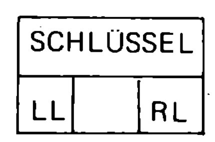
Die Abbildung 4.7 zeigt einen geordneten, ausbalancierten, binären Baum. Auf einer Diskette sind die Datensätze für die Vornamen in der Abbildung 4.8 gezeigten Reihenfolge gespeichert. Die Vornamen sind dabei die Schlüsselworte, nach denen der binäre Baum aufgebaut wurde. Jeder Knoten enthält neben dem Schlüsselwort noch zwei Zeiger RL und LL. Der Zeiger RL weist auf das nächst größere Schlüsselwort, das rechts vom betrachteten Knoten liegt. LL weist auf das nächst kleinere Element hin. Den Aufbau eines Knotens zeigt Abbildung 4.9.



4.7 Binärer Baum

1 EGON
2 DIETER
3 ANNA
4 CARLA
5 FRANZ
6 GERDA
7 BERTA

4.8 Namensliste



4.9 Knoten

Das oberste Element des Baumes ist das Schlüsselwort *Dieter*. Auf diesen Knoten zeigt ein Zeiger P, der den Stamm des Baumes markiert. P=2 zeigt, daß *Dieter* an zweiter Stelle auf der Diskette gespeichert ist. Der Zeiger RL=5 zeigt auf *Franz*, der Zeiger LL=7 auf *Berta*. Von *Berta* wird auf *Carla* (RL=4) und *Anna* (LL=3) verwiesen. Beide Namen stellen das Ende der Verzweigungen dar. Die Zeiger RL und LL in beiden Knoten sind Null.

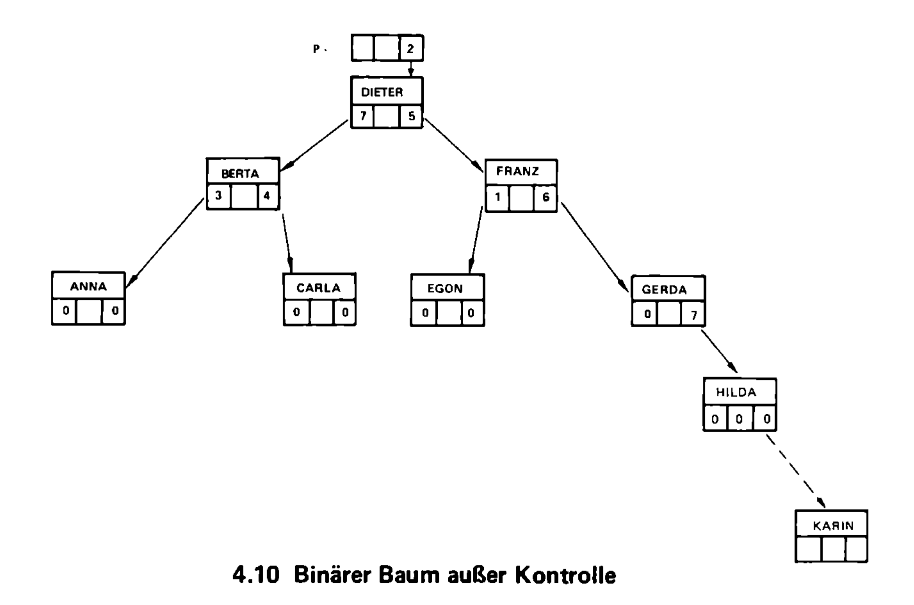
Wird nun ein Name gesucht, so wird der erste Vergleich mit P=2 begonnen und nach rechts oder links weitergegangen, bis der gesuchte Name gefunden ist. Wird der Name *Egon* gesucht, so wird beim ersten Vergleich festgestellt, daß *Egon* größer (in alphabetischer Reihenfolge) als *Dieter* ist. Beim Vergleich mit *Franz* ist *Egon* kleiner und beim nächsten Vergleich ist *Egon* gefunden.

Der Zeiger LL, der auf *Egon* zeigt, ist 1. Somit ist *Egon* der erste Datensatz auf der Diskette. Insgesamt waren drei Vergleiche notwendig, bis der Name gefunden war. Bei 15 Namen sind es 4, bei 21 Namen 5 usw. Soll aus 4095 Datensätze ein Name gesucht werden, so sind maximal 12 Vergleiche notwendig. Wird aber die Anzahl der Datensätze auf 8191 verdoppelt, so ist nur ein weiterer Vergleich (insgesamt dann 13) notwendig. Dies setzt aber voraus, daß der binäre Baum regulär und ausbalanciert ist. Auf diese Problematik soll gleich eingegangen werden.

Wo können solche Baumstrukturen verwendet werden?
Sie können überall dort verwendet werden, wo ein Schlüsselwort nicht zweimal vorkommen kann. Dies ist zum Beispiel bei Inventarverzeichnissen der Fall, wo eine Inventarnummer keine zwei verschiedenen Einträge bezeichnen kann. Für Bankleitzahlen oder für Versicherungsnummern kann diese Struktur ebenfalls verwendet werden. Auch Adressverwaltungen können aufgebaut werden. Der Herr Peter Fischer aus Rastatt wird vom Herrn Peter Fischer aus Ulm dadurch unterschieden, daß seinem Namen ein beliebiges Zeichen angehängt wird, das beim Ausdruck von Adressen unterdrückt wird.

Von dieser Art von binären Bäumen muß man andere Arten von Bäumen unterscheiden, wie sie bei Suchbäumen oder Expertensystemen aufgebaut werden. Bei diesen wird in einem Knoten nach Eigenschaften verzweigt. Zum Beispiel: Bluttemperatur größer 37.8° dann rechts weiter, weil krank, sonst links weiter, weil gesund.

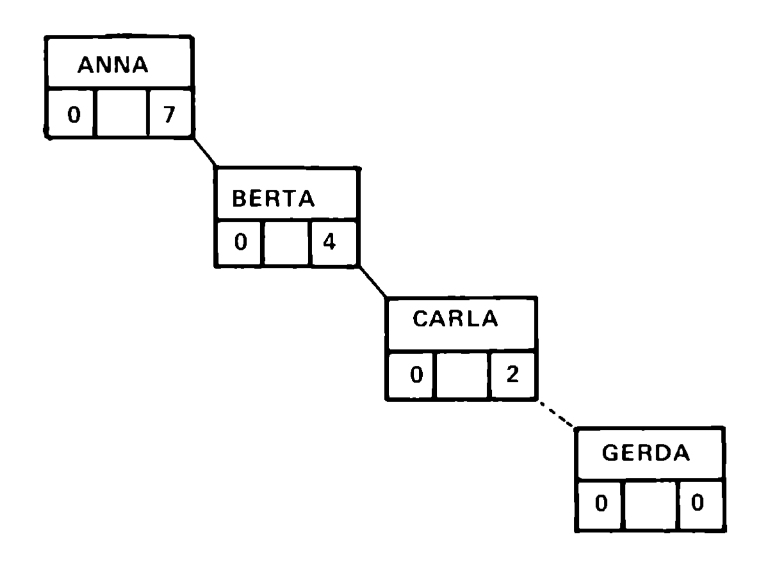
Nun wieder zurück zu unserem binären Baum. In diesen soll ein neuer Name *Hilde* eingefügt werden. Der Platz dafür wird nach *Gerda* gefunden. Soweit ist noch alles in Ordnung. Der nächste Name *Karin* würde seinen Platz nach *Hilde* erhalten. Unser schöner, binärer Baum gerät außer Kontrolle. Dies zeigt Abbildung 4.10. Der Aufbau eines Baumes ist also im wesentlichen durch die Reihenfolge der Eingabe der Schlüsselworte bestimmt. Wären die Namen in alphabetischer Reihenfolge eingegeben worden, so wäre ein Baum nach Abbildung 4.11 entstanden.



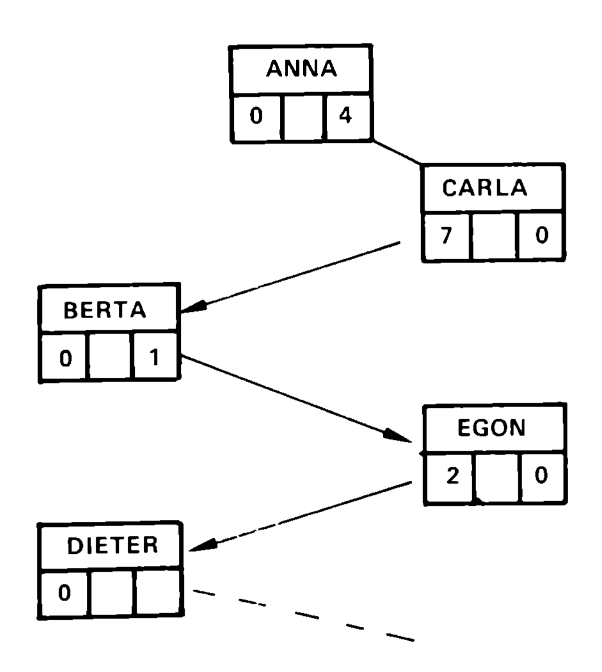
4.10 Binärer Baum außer Kontrolle

Soll hier ein Name gesucht werden, so müssen maximal sieben, statt drei Vergleiche durchgeführt werden. Nicht besser ist es mit einem Zick-Zack-Baum in Abbildung 4.12.

Solche Baumstrukturen sind selten und im allgemeinen ist ein binärer Baum bei einer zufälligen Eingabe von Schlüsselworten in den meisten Fällen einigermaßen ausbalanciert. Neben anderen Verfahren um binäre Bäume auszubalancieren, haben zwei russische Mathematiker Andelson-Velskii und Landis ein Verfahren angegeben, das einen binären Baum bei der Eingabe ausbalanciert hält. Das Verfahren für diese AVL-Bäume soll nun gezeigt werden.



4.11 Entarteter Baum

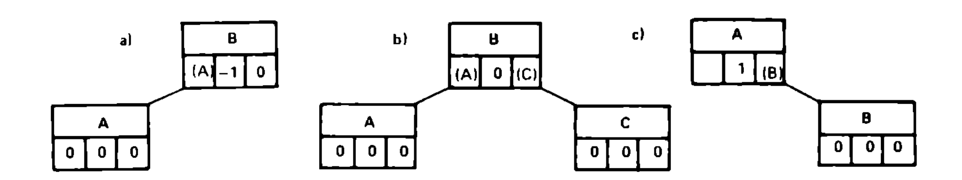


4.12 Zick-Zack Baum

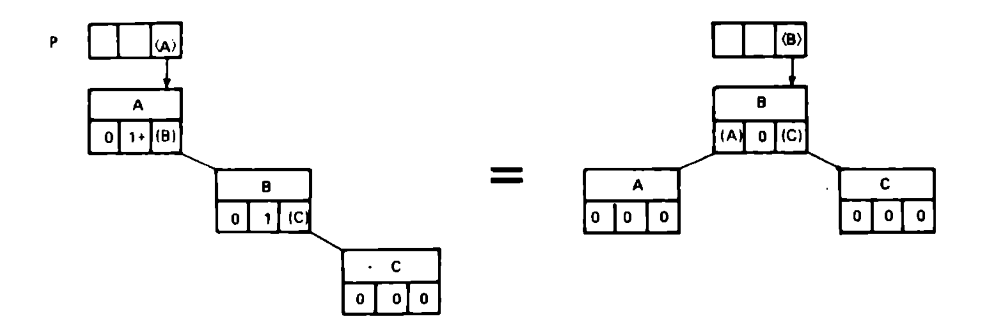
Zu den beiden Zeigern RL und LL wird eine weitere Information, der Faktor B hinzugefügt. Dieser Faktor gibt an, ob ein Baum nach rechts oder links wächst, oder ob er ausbalanciert ist.

B=1 bedeutet, daß der Baum nach rechts, B=—1, daß er nach links um eine Höhenstufe gewachsen ist. B=0 bedeutet, daß der Baum, oder ein Teil des Baumes, ab dieser Stelle nach rechts oder links gleich groß ist. B ist ebenfalls Null bei den Enden des Baumes.

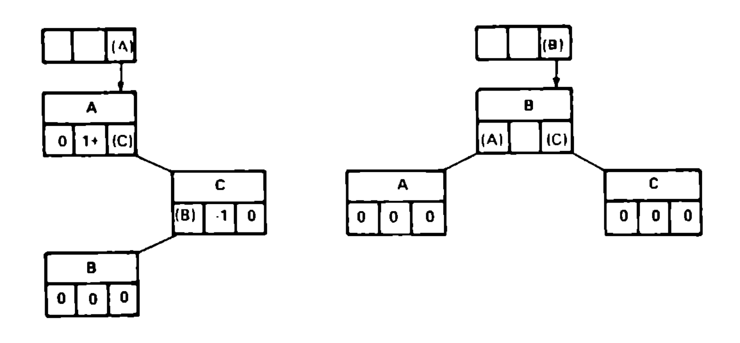
In den folgenden Beispielen wird als Schlüsselwort ein Buchstabe verwendet. Betrachten wir nun Fall 4.13. Dort soll ein neuer Knoten C eingefügt werden. Der Platz dafür wird rechts von B gefunden. Da der Baum nach rechts gewachsen ist, muß B(B)=1 werden. Ebenso B(A). Dies ist aber schon Eins. Dies zeigt, daß der Baum entartet und somit neu ausbalanciert werden muß.



4.13 Die drei möglichen Fälle für den Faktor B



4.14 Ausbalancieren eines entarteten Baumes. Einfache Verschiebung



4.15 Ausbalancieren eines entarteten Baumes. Zweifache Verschiebung

In diesem Fall wird eine einzige Verschiebung durchgeführt. B wird der neue Stamm. Nach LL(B) wird die Adresse von A geschrieben, RL(A) wird Null und der Anfangszeiger P enthält die Adresse von B. Die Schreibweise (B) in der Zeichnung soll anzeigen, daß nicht das Schlüsselwort A, sondern die Adresse von A gespeichert ist. Die gleiche Vertauschung wird bei einem Baum angewendet, der zweimal nach links gewachsen ist. Hier sind dann die Zeiger RL und LL vertauscht. In beiden Fällen entsteht ein ausbalancierter Baum und alle Faktoren B sind Null.

Einen zweiten Fall zeigt Abbildung 4.15. Der Baum enthält die Elemente A und C. Eine neues Element B wurde hinzugefügt. Von C aus gesehen, ist der Baum nach links gewachsen B(C)=–1.

Von A aus aber nach rechts. Da aber B(A)=1 ist, muß neu ausbalanciert werden. In diesem Fall ist eine zweimalige Verschiebung notwendig. Da B 〉 A aber B 〈 C ist, muß B zwischen A und C eingefügt und somit neuer Stamm werden. Die Zeiger der drei Elemente müssen entsprechend vertauscht werden. Entsprechendes gilt für einen Baum, der nach links und dann nach rechts gewachsen ist.

Im Programm wird der gleiche Aufbau des Datensatzes verwendet, wie bei den verketteten Listen. Das Schlüsselwort ist allerdings nur 25 Bytes lang, da für die zusätzliche Information zum Ausbalancieren ein weiteres Byte benötigt wird. Die Zeiger werden mit LL und RL bezeichnet.

Der Algorithmus ist nach (1) programmiert. Dieser wird im folgenden hier wiedergegeben. Anfängliche Schwierigkeiten bei der Programmierung wurden durch die Angabe, der Baum sei "nicht leer" hervorgerufen. Ein Baum ist dann nicht leer, wenn mindestens ein Element vorhanden ist. Der Algorithmus benötigt aber am Anfang einen "jungen Baum", d. h. einen Baum mit einem Stamm und zwei Ästen. Außerdem muß dieser Baum ausbalanciert sein. Dieser Baum wird durch das Wort Y-TREE bei der Eingabe aufgebaut.

Hierbei sind folgende Fälle zu unterscheiden:

1. Der Baum ist leer. Dann ist:
   PA=0

2. Ein Element. Dann ist:
   PA=1; LL(1)=0; RL(1)=0; B=0

3. Zwei Elemente. Dann ist:
   a) KW(2) ⟨KW(1)   KW = Schlüsselwort
      PA=1; LL(1)=2; RL(1)=0; B(1)=–1;
      LL(2)=0; RL(2)=0; B(2)=0;

   b) KW(2) ⟩KW(1)
      PA=1; LL(1)=0; RL(1)=2; B(1)=1;
      LL(2)=0; RL(2)=0; B(2)=0.

4. Drei Elemente: Dann ist:
   a) KW(3) ⟨KW(1) und LL(1)=0
      LL(1)=3; B(1)=0;

   b) KW(3) ⟩KW(1) und RL(1)=0
      RL(1)=3; B(1)=0;

   c) KW(3) ⟩KW(2) ⟩KW(1) (einfache Drehung)
      PA=2; LL(2)=1; RL(2)=3
      alle anderen Zeiger Null.

   d) KW(3) ⟨KW(2) ⟨KW(1) (einfache Drehung)
      PA=2; LL(2)=3; RL(2)=1; B(2)=0;
      alle anderen Zeiger Null.

   e) KW(3) ⟨KW(2) ⟩KW(1) (zweifache Drehung)
      PA=3; LL(3)=1; RL(3)=2; B(3)=0;
      alle anderen Zeiger Null.

f) KW(3) >KW(2) <KW(1) (zweifache Drehung)
PA=3; LL(3)=2; RL(3)=1; B(3)=0;
alle anderen Zeiger Null.

Das Wort 3EL fügt das dritte Element nach den oben angegebenen Bedingungen ein. Der besseren Übersicht ist dieses Wort nochmals in Abbildung 4.16 dargestellt.

```
: 3EL

 1 1 COMP DUP 0 =
         IF MES1 2DROP 0
         ELSE 0 >
           IF 1 @LL 0=
             IF 3 1 !LL 0 1 !B
             ELSE 2 COMP DUP 0=
               IF MES1 2DROP 0
               ELSE 0 >
                 IF 2 !PA 3 2 !LL 1 2 !RL 0 1 !B
                 ELSE 3 !PA 1 !NRL 2 !NLL 0 !NB 2 ZERO
                 THEN 1 ZERO
               THEN
             THEN
           ELSE 1 @RL 0 =
             IF 3 1 !RL 0 1 !B
             ELSE 2 COMP 0<
               IF 2 !PA 1 2 !LL 3 2 !RL 0 2 !B
               ELSE 3 !PA 1 !NLL 2 !NRL 0 !NB 2 ZERO
               THEN 1 ZERO
             THEN
           THEN
         THEN
           IF !MEM
           THEN     ;
```

4.16 Das Wort 3EL

Nachdem ein balancierter Baum aus drei Elementen aufgestellt wurde, kann der Algorithmus angewendet werden. Es werden im Programm die in (1) angegebenen Variablen verwendet.

Im ersten Teil des Algorithmus wird der Platz für ein neues Element gesucht.

A1. Anfang S = P = PA. T = 0

Die Zeigervariable P wandert am Baum entlang. Die Zeigervariable S zeigt auf die Stelle, bei der eine Ausbalancierung notwendig wird. T zeigt jeweils auf den Vater von S.

A2. Vergleich: Wenn KW(N) ⟨ KW(P), dann weiter mit A3. Wenn KW(N) ⟩ KW(P), dann weiter mit A4. Wenn KW(N) = KW(P), dann wird die Platzsuche abgebrochen.

A3. Nach links. Q=LL(P). Wenn Q leer ist, wird Q=N und LL(P)=Q und weiter mit A5. Sonst wird, wenn B(Q) ⟨⟩ 0 ist, T=P und S=Q. Dann wird P=Q und nach A2 zurückgegangen.

A4. Nach rechts. Q=RL(P). Wenn Q leer ist, wird Q=N und LL(P)=Q und weiter mit A5. Sonst wird, wenn B(Q) ⟨⟩ 0 ist, T=P und S=Q. Dann wird P=Q und nach A2 zurückgegangen.

A5. Einsetzen. Das neue Element wird gespeichert. Alle Zeiger dieses Elementes sind Null.

Dieser Teil des Algorithmus wird durch das Wort SEARCH-PLACE in Abbildung 4.17 ausgeführt.

Als nächstes müssen die Zeiger B neu gesetzt werden.

A6. Setzen der Zeiger B zwischen S und Q. Wenn KW(N) ⟨ KW(S) dann wird P und U gleich LL(S), sonst wird P und U gleich RL(S). Dann wird B(P)=–1, wenn KM(N) ⟨ KW(P) oder B(P)=+1, wenn KW(N) ⟩ KW(P) ist.
Wenn P=Q ist, dann ist dieser Teil beendet.

Nun wird festgestellt, ob der Baum neu ausbalanciert werden muß. Dazu wird die Variable A benötigt.

```
: SEARCH-PLACE

   @PA DUP S ! P ! 0 T !
   BEGIN 1 P @ COMP DUP 0=
     IF MES1 2DROP 0 0
     ELSE 0 >
       IF P @ @LL DUP Q ! 0=
         IF @N DUP Q ! P @ !LL DROP 1 0
         THEN
       ELSE P @ @RL DUP Q ! 0=
         IF @N DUP Q ! P @ !RL DROP 1 0
         THEN
       THEN
     THEN
   WHILE Q @ @B 0= NOT
     IF P @ T ! Q @ S !
     THEN Q @ P !
   REPEAT
     IF !MEM
     THEN    ;
```

4.17 Das Wort SEARCH-PLACE

A7. Wenn KW(N) < KW(S) ist, dann wird A gleich −1, sonst +1.

Wenn B(S) gleich −A ist, dann wird B(S) gleich Null. Der Baum ist ausbalanciert und der Algorithmus ebenfalls beendet.

Wenn B(S) gleich A ist, dann muß der Baum neu ausbalanciert werden. Wenn B(U) gleich A ist, dann wird in A8 eine einfache Drehung durchgeführt, sonst wird in A9 eine doppelte Drehung ausgeführt.

A8. Einfache Drehung.
Wenn A > 0 ist, dann einfache Drehung nach rechts mit:
P=R; RL(S)=LL(U); LL(U)=S,
sonst einfache Drehung nach links mit:
P=R; LL(S)=RL(U); RL(U)=S.
Dann wird B(S)=B(U)=0.

A9. Doppelte Drehung.
Wenn A > 0 ist, dann doppelte Drehung nach rechts mit:
P-LL(U); LL(U)=RL(P); RL(P)=U;
RL(S)-LL(P); LL(P)=S,
sonst doppelte Drehung nach links mit:
P-RL(U); RL(U)=LL(P); LL(P)=U;
LL(S)=RL(P); RL(P)=S.
Wenn B(P)=A ist, dann B(S)=—A; B(U)=0.
Wenn B(P)=—A ist, dann B(U)=—A; B(S)=0.
Wenn B(P)=0 ist, dann B(S)=B(U)=0.

A10. Ende des Ausbalancierens:
Wenn S=RL(T), dann RL(T)=P, sonst LL(T)=P.

Das Ausbalancieren des Baumes wird im Programm durch das Wort BALANCE durchgeführt. Das Programm zeigt Abbildung 4.18. Ein Beispiel ist in Abbildung 4.19 angegeben.

```
SCR # 40
  0 ( AVL                          10.3.84 EF)
  1 1 CONSTANT START
  2 32 CONSTANT RECLEN
  3 : EXIT COMPILE ;S ; IMMEDIATE
  4
  5 : @PA ( -N) START BLOCK 26 + @ ;
  6  : !PA ( N) START BLOCK 26 + !
  7   UPDATE ;
  8 : #INDEX ( N-A) RECLEN B/BUF
  9   */MOD START + BLOCK + ;
 10
 11 : @LL ( N-N') #INDEX 26 + @ ;
 12 : @RL ( N-N') #INDEX 28 + @ ;
 13 : @N ( -N) 0 #INDEX @ ;
 14 -->
 15
```

```
SCR # 41
  0 ( AVL CNTD              10.3.84 EF)
  1 : !NLL ( N) PAD 26 + ! ;
  2 : !NRL ( N) PAD 28 + ! ;
  3 : !NB ( N) PAD 25 + C! ;
  4 : !LL ( NN') #INDEX 26 + !
  5   UPDATE ;
  6 : !RL ( NN') #INDEX 28 + !
  7   UPDATE ;
  8 : !N ( N) 0 #INDEX ! UPDATE ;
  9 : +N 1 0 #INDEX +! UPDATE ;
 10 : @B ( N-N') #INDEX 25 + C@
 11 DUP 255 = IF DROP -1 THEN ;
 12 : !B ( NN') #INDEX 25 + C!
 13   UPDATE ;
 14 : MES1 ." SCHON IN LISTE" ;
 15 -->
```

```
SCR # 42
  0 ( AVL COMPARE           10.3.84 EF)
  1 : COMPARE ( AA'N-F) ( F -0+ )
  2   OVER + SWAP
  3   DO COUNT I C@ - -DUP
  4    IF SWAP 0= LEAVE THEN
  5   LOOP IF 0 THEN ;
  6
  7 : COMP ( N-F) #INDEX PAD 25
  8   COMPARE ;
  9
 10 -->
 11
 12
 13
```

```
SCR # 43
  0 ( AVL CNTD               10.3.84 EF)
  1 :  !MEM PAD @N DUP PAD 30 + !
  2 #INDEX RECLEN CMOVE UPDATE +N ;
  3
  4 : EMPTY-TREE 0 !PA 1 !N ;
  5 : ZERO ( N) 0 SWAP 2DUP 2DUP
  6    !LL !RL !B ;
  7
  8 : @MEM ( N) #INDEX PAD RECLEN
  9   CMOVE ;
 10
 11 -->
 12
 13
 14
 15
```

```
SCR # 44
  0 ( AVL Y-TREE             10.3.84 EF)
  1 : 1EL 1 !PA 0 0 0 !NLL !NRL !NB
  2    !MEM ;
  3
  4 : 2EL 1 COMP DUP 0= IF MES1 DROP
  5   ELSE 0< IF 2 1 !RL 1 1 !B
  6     ELSE 2 1 !LL -1 1 !B THEN
  7   1EL THEN ;
  8
  9
 10
 11
 12
 13
 14 -->
 15
```

```
SCR # 45
  0 ( AVL CNTD                  10.3.84 EF)
  1 : 3EL 1 1 COMP  DUP 0=
  2 IF MES1 2DROP 0 ELSE 0 >
  3  IF 1 @LL 0= IF 3 1 !LL 0 1 !B
  4   ELSE 2 COMP DUP 0= IF MES1
  5   2DROP 0 ELSE 0 >
  6 IF 2 !PA 3 2 !LL 1 2 !RL 0 2 !B
  7 ELSE  3 !PA 1 !NRL 2 !NLL 0 !NB
  8       2 ZERO THEN 1 ZERO THEN
  9   THEN ELSE 1 @RL 0=
 10   IF 3 1 !RL 0 1 !B ELSE
 11   2 COMP 0< IF 2 !PA 1 2 !LL
 12   3 2 !RL 0 2 !B ELSE
 13 3 !PA 1 !NLL 2 !NRL 0 !NB 2 ZERO
 14  THEN 1 ZERO THEN THEN THEN
 15  IF !MEM ELSE QUIT THEN ;  -->
```

```
SCR # 46
  0 ( AVL Y-TREE                10.3.84 EF)
  1 : Y-TREE @PA 0= IF 1EL ELSE
  2   @N 2 > IF 3EL ELSE
  3   2EL THEN THEN ;
  4
  5 : PADC PAD 25 32 FILL 0 0 0 !NLL
  6    !NRL !NB ;
  7
  8 0 VARIABLE T 0 VARIABLE S
  9 0 VARIABLE P 0 VARIABLE Q
 10 0 VARIABLE U 0 VARIABLE A
 11 -->
 12
 13
 14
 15
```

```
SCR # 47
  0 ( AVL SEARCH          10.3.84 EF)
  1 : SEARCH-PLACE
  2 @PA DUP  S ! P ! 0 T !
  3 BEGIN 1 P @ COMP DUP 0=
  4 IF  MES1 2DROP 0 0 ELSE 0 >
  5   IF P @ @LL DUP Q ! 0=
  6 IF @N DUP Q ! P @ !LL DROP 1 0
  7 THEN ELSE P @ @RL DUP Q ! 0=
  8 IF @N DUP Q ! P @ !RL DROP 1 0
  9   THEN THEN THEN
 10 WHILE
 11  Q @ @B 0= NOT
 12   IF P @ T ! Q @ S ! THEN
 13  Q @ P !
 14 REPEAT IF !MEM ELSE QUIT THEN ;
 15 -->
```

```
SCR # 48
  0 ( AVL ADJUST          10.3.84 EF)
  1 : ADJUST S @ DUP COMP 0<
  2 IF @RL ELSE @LL THEN DUP U ! P !
  3  BEGIN
  4  P @ DUP DUP COMP DUP 0= NOT
  5  WHILE 0<
  6  IF 1 SWAP !B @RL ELSE
  7  -1 SWAP !B @LL THEN P !
  8  REPEAT 2DROP ;
  9
 10 : A8 A @ 0 > U @ P !
 11  IF U @ @LL S @ !RL S @ U @ !LL
 12 ELSE U @ @RL S @ !LL S @ U @ !RL
 13  THEN 0 S @ !B 0 U @ !B ;
 14 -->
 15
```

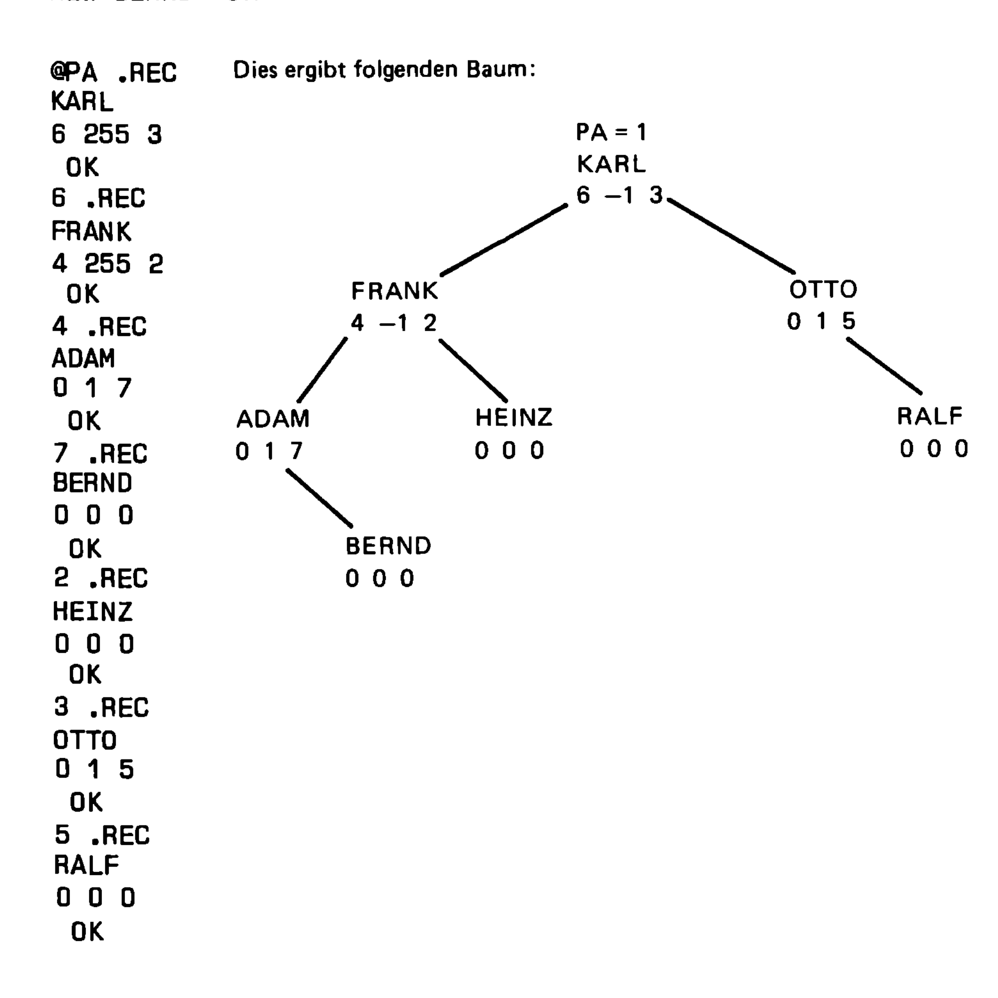
```
SCR # 49
  0 ( AVL BALANCE          10.3.84 EF)
  1 : A9 A @ 0 > IF
  2  U @ @LL P ! P @ @RL U @ !LL
  3  U @ P @ !RL
  4  P @ @LL S @ !RL S @ P @ !LL
  5 ELSE
  6  U @ @RL P ! P @ @LL U @ !RL
  7  U @ P @ !LL
  8  P @ @RL S @ !LL S @ P @ !RL
  9 THEN P @ @B DUP 0=
 10 IF 0 S @ !B 0 U @ !B DROP
 11 ELSE A @ + 0=
 12 IF
 13 0 S @ !B A @ -1 * U @ !B ELSE
 14 0 U @ !B A @ -1 * S @ !B
 15 THEN THEN 0 P @ !B ; -->
```

```
SCR # 50
  0 ( AVL BALANCE          10.3.84 EF)
  1 : BALANCE
  2  S @ COMP 0< IF 1 ELSE -1 THEN
  3  A ! S @ @B DUP 0=
  4 IF DROP A @ S @ !B
  5 ELSE A @ + 0=
  6     IF 0 S @ !B
  7     ELSE U @ @B A @ +
  8      IF A8 ELSE A9 THEN
  9 T @ @RL S @ = >R P @ T @ R>
 10 IF !RL ELSE !LL THEN
 11     THEN
 12 THEN ;
 13 -->
 14
 15
```

```
SCR # 51
  0 ( AVL CNTD              10.3.84 EF)
  1 : NEW PADC 13 WORD HERE COUNT
  2   PAD SWAP CMOVE @N 3 >
  3   NOT IF Y-TREE ELSE
  4   SEARCH-PLACE ADJUST BALANCE
  5   THEN ;
  6
  7 : .REC ( N) CR @MEM PAD 24 TYPE
  8   CR PAD 26 + @ . PAD 25 + C@ .
  9   PAD 28 + @ . CR ;
 10
 11
 12
 13
 14
 15
```

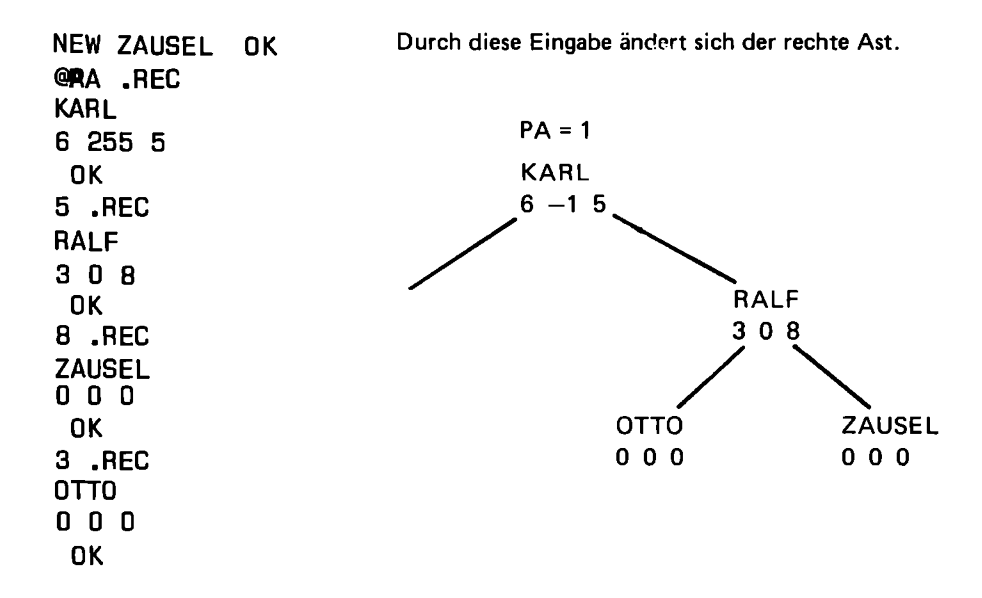
4.18 Aufstellen von binären Bäumen mit Ausbalancieren

```
EMPTY-TREE  OK
NEW KARL  OK
NEW HEINZ  OK
NEW OTTO  OK
NEW ADAM  OK
NEW RALF  OK
NEW FRANK  OK
NEW BERND  OK
```



```
NEW ZAUSEL  OK
@PA .REC
KARL
6 255 5
 OK
5 .REC
RALF
3 0 8
 OK
8 .REC
ZAUSEL
0 0 0
 OK
3 .REC
OTTO
0 0 0
 OK
```

Durch diese Eingabe ändert sich der rechte Ast.



4.19 Ein Beispiel



Ein Beispiel für künstliche Intelligenz

Das folgende Programm wurde freundlicherweise von Herrn Dr. Schmitter zur Verfügung gestellt.

Unter dem Begriff "künstlicher Intelligenz" soll folgendes verstanden werden:

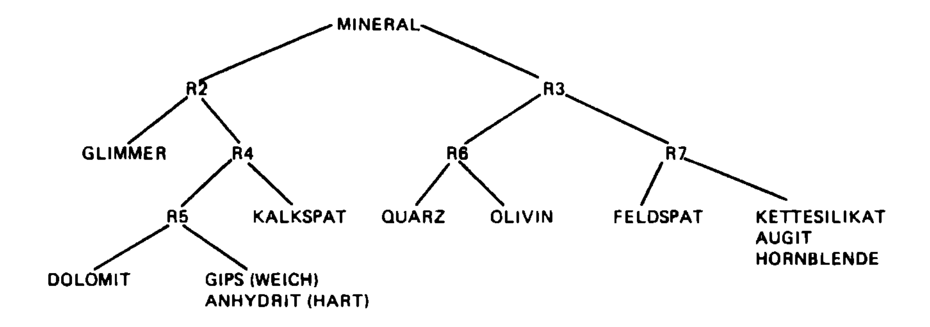
Werden von einer Maschine Aufgaben erledigt, die, wenn sie von einem Menschen ausgeführt würden, einen gewissen Intelligenzgrad voraussetzen, so soll dies als künstliche Intelligenz bezeichnet werden.

Zu diesen Aufgaben gehören zum Beispiel die Erkennung von Mustern, die Spracherkennung, automatische Programmerstellung, Lösung von Problemen, die nicht durch Algorithmen beschrieben werden können und die Programmierung von Expertensystemen.

Aus dem letzteren Bereich sollen zwei einfache Beispiele gezeigt werden. In den ersten beiden Beispielen wird durch Entscheidungsbäume die Bestimmung eines Minerals und eine Fehleranalyse durchgeführt, im dritten Beispiel werden Worte durch Assoziationen verknüpft. Dazu werden Eigenschaften des FORTH Kernels benutzt.

5.1 Entscheidungsbäume

Im Gegensatz zu den im letzten Kapitel beschriebenen binären Bäumen werden hier Entscheidungsbäume benutzt. Einen solchen Entscheidungsbaum zeigt Abbildung 5.1. Dieser dient zur Bestimmung von Mineralien.



5.1 Entscheidungsbaum zur Bestimmung von Mineralien

Zur Bestimmung werden bestimmte Begriffe und Eigenschaften festgelegt.

Der oberste Knoten ist der Härtegrad. Je nachdem, ob das zu bestimmende Mineral ritzbar oder nicht ist, wird nach links oder rechts im Baum verzweigt. Weitere Begriffe zur Bestimmung sind die Farbe, die Spaltbarkeit und die Löslichkeit. Als Eigenschaften werden für die Farbe zum Beispiel hell-rötlich oder grau-durchscheinend aufgeführt. Das Programm zeigt Abbildung 5.2.

```
SCR # 102
  0 ( AI                             DRS )
  1 0 VARIABLE OFFS
  2 : INIT 0 OFFS ! ;
  3 : BEGRIFF <BUILDS 12 ALLOT DOES>
  4   4 - NFA DUP C@ 127 - PAD OFFS
  5   @ + SWAP CMOVE 13 OFFS +! ;
  6
  7 : AUS. PAD OFFS @ + DUP C@ 128 -
  8   SWAP 1+ SWAP TYPE 13 OFFS +! ;
  9
 10 : ANTWORT ." ? (WAHR=1/FALSCH=0
 11 )" KEY 48 - DUP . CR ;
 12
 13 : NAME 4 - NFA DUP C@ 128 -
 14   SWAP 1+ SWAP TYPE ;
 15 -->
```

```
SCR # 103
  0 ( AI CNTD                        DRS )
  1 : AUSSAGE DOES> ."  " NAME ."  "
  2   ANTWORT ;
  3 : EIGEN1 <BUILDS
  4   DOES> INIT CR ."  IST "
  5   AUS. ."  " NAME ANTWORT INIT ;
  6
  7 : EIGEN2 DOES> INIT CR ."  IST "
  8   NAME ."  " AUS. ANTWORT INIT ;
  9
 10 : IMP SWAP NOT OR ;
 11 : ID = ;
 12
 13
 14 -->
 15
```

```
SCR # 104
  0 ( AI CNTD                        DRS )
  1 : BE BEGRIFF ; : E1 EIGEN1 ;
  2
  3 BE HAERTEGRAD
  4 BE BRUCH
  5 BE FARBE
  6 BE SPALTBARKEIT
  7 BE LOESLICHKEIT
  8 E1 AUFBRAUSEND
  9 E1 ALS-PULVER
 10 E1 PLATTIG
 11 E1 HELL-ROETLICH
 12 E1 GRAU-DURCHSCHEINEND
 13 E1 MUSCHELIG
 14 E1 RITZBAR
 15 -->
```

```
SCR # 105
  0 ( AI MINERAL                    DRS )
  1 : E HOME ." STARTWORT MINERAL "
  2   CR ; : 1= 1 = ;
  3 : R7 FARBE HELL-ROETLICH 1 = IF
  4 ." FELDSPAT " ELSE ." DUNKELBRAU
  5 N -> KETTENSILIKAT: AUGIT, HORNB
  6 LENDE " ENDIF ;
  7 : R6 FARBE GRAU-DURCHSCHEINEND
  8  1= IF ." QUARZ " ELSE ." GRUENL
  9 ICH -> OLIVIN" ENDIF ;
 10 : R5 LOESLICHKEIT ALS-PULVER 1=
 11   IF ." DOLOMIT " ELSE ." WEICH
 12 -> GIPS HART -> ANHYDRIT" THEN ;
 13  : R4 LOESLICHKEIT AUFBRAUSEND
 14 1= IF ." KALKSPAT" ELSE R5 THEN
 15  ; -->
```

```
SCR # 106
  0 ( AI MINERAL                    DRS )
  1 : R3 BRUCH MUSCHELIG 1= IF R6
  2   ELSE ." BRUCHFLAECHE -> EBEN "
  3   R7 THEN ;
  4 : R2 SPALTBARKEIT PLATTIG 1= IF
  5   ." GLIMMER (BLATTSILIKAT) "
  6 ELSE ." KOERNIG" R4 THEN ;
  7
  8 : MINERAL E HAERTEGRAD RITZBAR
  9   1= IF R2 ELSE R3 THEN ; E ;S
 10
 11
 12
 13
 14
 15
```

5.2 Bestimmung von Mineralien durch einen Entscheidungsbaum

Der in Abbildung 5.1 angegebene Baum wird in den Textfeldern 105 und 106 programmiert. Das Wort Mineral leitet die Bestimmung ein. Abbildung 5.3 zeigt ein Beispiel.

```
MINERAL STARTWORT MINERAL

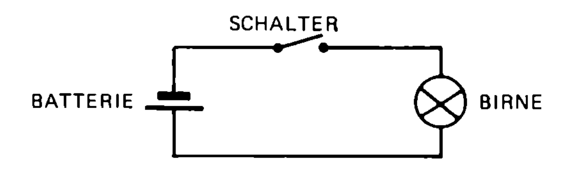
 IST HAERTEGRAD RITZBAR ? (WAHR=1/FALSCH=0)1

 IST SPALTBARKEIT PLATTIG ? (WAHR=1/FALSCH=0)0
KOERNIG
 IST LOESLICHKEIT AUFBRAUSEND ? (WAHR=1/FALSCH=0)1
KALKSPAT OK
```

5.3 Beispiel

Die Begriffe und die Eigenschaften werden im Textfeld 104 festgelegt. Dabei werden zwei Definitionsworte BEGRIFF und EIGEN1 verwendet. Mit dem Wort BEGRIFF werden Worte definiert, die beim Kompilieren für 12 Bytes Speicherplatz im Wörterbuch freihalten. Während der Laufzeit werden diese Worte nach PAD geschrieben, wobei eine Variable OFFS auf den augenblicklich verwendeten Begriff zeigt. Das zweite Definitionswort EIGEN1 legt während des Kompilierens nur den Kopf des Wortes an, und gibt während der Laufzeit den Namen des Entscheidungsbegriffes und die Eigenschaft aus. In Antwort wird auf eine Eingabe (1 = wahr, 0 = falsch) gewartet und dementsprechend verzweigt. Das Wort AUS. gibt den Begriff, das Wort NAME die Eigenschaft aus.

Das zweite Beispiel ist eine einfache Fehleranalyse. Dieses Programm benutzt die Worte aus den Textfeldern 102 und 103. Gegeben ist ein kleiner Schaltkreis, bestehend aus Schalter, Batterie und Birne (Abbildung 5.4).



5.4 Zu testender Schaltkreis

Den Begriffen Schalter, Birne und Batterie sind die Eigenschaften ZU., AN. und IN.ORDNUNG zugeordnet.

```
SCR # 107
  0 ( AI FEHLERANALYSE         DRS )
  1 BEGRIFF SCHALTER BEGRIFF BIRNE
  2 BEGRIFF BATTERIE
  3 EIGEN1 ZU. EIGEN1 AN.
  4 EIGEN1 IN.ORDNUNG
  5
  6 : PRUEFE BATTERIE IN.ORDNUNG
  7   0= IF ." BATTERIE WECHSELN"
  8    ELSE BIRNE IN.ORDNUNG
  9   0= IF ." BIRNE WECHSELN"
 10   ELSE ." SCHALTER SCHLIESST NIC
 11 HT" THEN THEN ;
 12
 13
 14 —>
 15
```

```
SCR # 108
  0 ( AI FEHLERANALYSE         DRS )
  1 : ANALYSE 2DUP SWAP IMP 0=
  2   IF ." SCHALTERKURZSCHLUSS"
  3  ELSE IMP 0= IF PRUEFE THEN CR
  4   ." TESTENDE" THEN ;
  5
  6 : TEST SCHALTER ZU. BIRNE AN.
  7  2DUP ID 1= IF 2DROP ." STARTE "
  8  ." MIT ANDERER SCHALTERSTELLUNG
  9 " ELSE ANALYSE THEN ; HOME
 10  ." TEST"
 11
 12
 13
 14
 15  ;S
```

```
0 PR#  OK
TEST
 IST SCHALTER ZU. ? (WAHR=1/FALSCH=0)1

 IST BIRNE AN. ? (WAHR=1/FALSCH=0)0

 IST BATTERIE IN.ORDNUNG ? (WAHR=1/FALSCH=0)1

 IST BIRNE IN.ORDNUNG ? (WAHR=1/FALSCH=0)1
SCHALTER SCHLIESST NICHT
TESTENDE OK
```

5.5 Fehleranalyse

Im ersten Testschritt wird die Zuordnung Birne und Schalter getestet. Ist zum Beispiel die Birne an und der Schalter geschlossen, so scheint alles in Ordnung zu sein. Dennoch muß ein weiterer Test mit der anderen Schalterstellung gemacht werden, ob die Birne wirklich ausgeht. Liegt wirklich ein Fehler vor, die Birne brennt nicht, obwohl der Schalter geschlossen ist, so wird der Fehler analysiert und durch PRUEFE die einzelnen Elemente überprüft. Das Programm zeigt Abbildung 5.5.

Dies ist natürlich nur ein einfaches Beispiel, das aber durchaus auf komplexe Schaltkreise übertragen werden kann. Verfügt der Rechner über geeignete Sensoren, so ist das Programm durchaus in der Lage, eine Funktionsprüfung des Schaltkreises selbsttätig durchzuführen, zum Beispiel in einem Auto zur Überwachung der Beleuchtungsanlage.

5.2. Assoziationen

Im vorherigen Beispiel mußten die Entscheidungsbäume fest programmiert werden. In Abbildung 5.6 ist ein Programm angegeben, das ein Aufstellen eines solchen Entscheidungsbaumes wesentlich vereinfacht. Das Wort BEGRIFF im Textfeld 108 hat eine andere Bedeutung, als das bisher verwendete Wort. Es ist ebenfalls ein Definitionswort.

```
SCR # 100
  0 ( AI ASSOZIATION            DRS )
  1 : BEGRIFF <BUILDS 21 ALLOT DOES>
  2   DUP 1 + SWAP C@ DUP 32 = IF
  3   ." NOCH NICHT ERKLAERT" DROP
  4   ELSE 0 DO DUP I 2 * + @ ID.
  5   LOOP ENDIF DROP ;
  6
  7 : ['] [CONPILE]  ' ;
  8 : [''] ['] ['] ;
  9
 10 : ZAHL DUP 2 + C@ DUP 32 = IF
 11   DROP 0 ENDIF ;
 12
 13 : ERHOEHE 1 + 2DUP SWAP 2 + C! ;
 14
 15 —>
```

```
SCR # 101
  0 ( AI ASSOZIATION            DRS )
  1
  2 : SCHREIB 2 * 1 + + SWAP NFA
  3   SWAP ! ;
  4
  5 : -> ['']ZAHL DUP 10 = IF
  6   ." FELD VOLL " DROP 2DROP
  7   ELSE ERHOEHE SCHREIB ENDIF ;
  8
  9
 10
 11
 12
 13
 14
 15 ;S
```

5.6 Assoziationen

Während des Kompilierens reservieren, damit definierte Worte 21 Bytes Speicherplatz im Wörterbuch. In diesen freien Platz wird durch das Wort – 〉 die Namensfeldadresse eines anderen Begriffs gespeichert.

Gleichzeitig wird in der ersten Stelle die Zahl der zugeordneten Begriffe gespeichert. Wird während der Laufzeit ein Begriff aufgerufen, so werden alle zugeordneten Begriffe, deren Namensfeldadresse gespeichert ist, ausgedruckt. Ist kein Begriff zugeordnet, so erscheint die Meldung "NOCH NICHT ERKLAERT". Das folgende Beispiel: "was mache ich am Abend" soll dies verdeutlichen.

Die folgenden Begriffe werden eingegeben:

```
SCR # 147
  0 ( ASSOCIATIONEN  BEISPIEL        )
  1 BEGRIFF ABEND   BEGRIFF DISCO
  2 BEGRIFF KINO    BEGRIFF THEATER
  3 BEGRIFF RESTAURANT
  4 BEGRIFF AUSGEHEN BEGRIFF ZUHAUSE
  5  BEGRIFF FERNSEHEN BEGRIFF LESEN
  6  BEGRIFF MUSIK BEGRIFF STRICKEN
  7 BEGRIFF SCHLAFEN BEGRIFF TANZEN
  8 BEGRIFF ESSEN BEGRIFF TRINKEN
  9 BEGRIFF WESTERN BEGRIFF KRIMI
 10 BEGRIFF SHOW
 11
```

Dann werden folgende Zuordnungen gemacht:

```
SCR # 148
  0 ( ASSOCIATIONEN BEISPIEL         )
  1 -> AUSGEHEN  ABEND
  2 -> ZUHAUSE  ABEND
  3 -> DISCO  AUSGEHEN
  4 -> KINO  AUSGEHEN
  5 -> THEATER  AUSGEHEN
  6 -> RESTAURANT  AUSGEHEN
  7 -> FERNSEHEN  ZUHAUSE
  8 -> LESEN  ZUHAUSE
  9 -> STRICKEN  ZUHAUSE
 10 -> SCHLAFEN  ZUHAUSE
 11 -> WESTERN  KINO
 12 -> WESTERN  FERNSEHEN
 13 -> KRIMI  KINO
 14 -> KRIMI  FERNSEHEN
 15
```

Ein Beispiel:

```
ABEND AUSGEHEN ZUHAUSE  OK
ZUHAUSE FERNSEHEN LESEN STRICKEN SCHLAFEN  OK
FERNSEHEN WESTERN KRIMI  OK
WESTERN NOCH NICHT ERKLAERT OK
```

6 Hilfen für die Programmentwicklung

FORTH Programme lassen sich im allgemeinen recht leicht auf Programmierfehler untersuchen. Jedes Wort kann einzeln getestet und der Inhalt des Stapels mit .S auf dem Bildschirm angezeigt werden. Manchmal ist die Fehlersuche trotzdem schwierig, vor allem dann, wenn ein logischer Fehler bei der Programmerstellung gemacht wurde. Dann hilft unter Umständen ein Breakpoint.

6.1 Breakpoint

(aus FORTH-DIMENSIONS V5, H1, S19)

```
( BREAKPOINT                        )

: BREAK CR ." S= " .S
BEGIN QUERY INTERPRET ." AOK"
CR AGAIN ;

: GO R> DROP R> DROP ;
```

6.1 Breakpoint

Abbildung 6.1 enthält einen, gegenüber der Originalveröffentlichung, leicht vereinfachten Breakpoint. Das Wort BREAK gibt den Inhalt des Stapels auf den Bildschirm aus und geht dann in eine unendliche Schleife. In dieser wird QUERY aufgerufen. In diesem Wort wartet das System auf eine Eingabe. Diese wird durch INTERPRET ausgeführt. Nun ist man sozusagen in einer höheren Ebene im FORTH-System.

Hier wird durch AOK angezeigt, daß eine neue Eingabe erwartet wird. Wird dabei die Fehlerroutine durchlaufen, so wird in dieser das Wort QUIT ausgeführt und man ist wieder in der alten Ausgangsebene. Sind beim Testen in der höheren Ebene die beiden Stapel nicht verändert worden, so kann mit GO das Programm weitergeführt werden.

6.2 Ein komfortabler Decompiler

(von Klaus-Peter Berg und Horst Warnecke aus CAL 1/84 S. 46)

Obwohl man meistens die Worte in einem Textfeld nachlesen kann, ist es manchmal doch sinnvoll, ein definiertes Wort zu decompilieren. Dies ermöglicht der Decompiler in Abbildung 6.2. In der Originalveröffentlichung ist dieser in FORTH-79 geschrieben. Dies ist eine Fassung für Fig-FORTH.

Das Wort ARRAY, das in Fig-FORTH nicht enthalten ist, ist ein Definitionswort. Wird durch ARRAY ⟨NAME⟩ ein Eintrag in das Wörterbuch gemacht, so muß die Anzahl der zu speichernden Zahlen angegeben werden.

```
SCR # 120
  0 ( FAW DECOMPILER BERG/WARNECKE )
  1 : ARRAY <BUILDS 2 * ALLOT DOES>
  2   SWAP 2 * +  ;
  3 : GETP [COMPILE] ' ;
  4 0 VARIABLE IFS 15 ARRAY IFT
  5 : CRO CR IFS @ 2+ SPACES ;
  6 : BR CRO DROP DUP 2+ DUP @ DUP
  7   0< IF ." REPEAT" + @ ." ( "
  8   2+ NFA ID. ." ) " ELSE
  9 ." ELSE " DROP DROP THEN 2+ 0 ;
 10
 11
 12
 13 -->
 14
 15
```

```
SCR # 121
  0 ( FAW DECOMPILER CNTD            )
  1 : OBR
  2 CRO DROP DUP 2+ DUP @ DUP 0< IF
  3 ." UNTIL " + @ ." ( " 2+ NFA ID.
  4  ." ) " ELSE + 4 - @ ' BRANCH
  5  CFA = IF DUP 2+ DUP @ + 2 - @
  6  0< IF ." WHILE " ELSE ." IF "
  7  DUP 2+ DUP @ + 2 - DUP @ + 2 -
  8  1 IFS +! IFS @ IFT ! THEN ELSE
  9 ." IF " DUP 2+ DUP @ + 2 - 1
 10  IFS +! IFS @ IFT ! THEN THEN
 11 2+ 0 ;
 12
 13 126 LOAD
 14
 15
```

```
SCR # 126
  0 ( DECOMPILER CNTD                )
  1 : LP BEGIN
  2     BEGIN DUP
  3     IFS @ IFT @ =
  4     WHILE CRO -1 IFS +!
  5    ." THEN " REPEAT 2+ DUP @
  6  DUP ' COMPILE CFA =
  7   IF CRO 2 SPACES 2+ NFA ID.
  8   2+ DUP @ 2+ NFA 0 THEN
  9  DUP ' LIT CFA =
 10   IF DROP DUP DUP 4 + @
 11   2+ ' CFA =
 12    IF ." ' " 2+ @ NFA ID.
 13    ELSE 2+ @ . THEN
 14   2+ 0 THEN
 15 -->
```

```
SCR # 127
  0 ( DECOMPILER CNTD                      )
  1  DUP ' BRANCH CFA =
  2   IF BR THEN
  3  DUP ' OBRANCH CFA =
  4   IF OBR THEN
  5  DUP ' (LOOP) CFA =
  6   IF DROP ." LOOP " 2+ 0 THEN
  7  DUP ' (+LOOP) CFA =
  8   IF DROP ." +LOOP " 2+ 0 THEN
  9  DUP ' (DO) CFA =
 10   IF DROP ." DO " 0 THEN
 11  DUP ' (.") CFA =
 12   IF DROP ." ." 34 EMIT SPACE
 13   2+ DUP COUNT DUP >R TYPE R>
 14   1 - + 34 EMIT SPACE 0 THEN
 15 -->
```

```
SCR # 128
  0 ( DECOMPILER CNTD                      )
  1 DUP
  2  IF
  3  DUP ' (;CODE) CFA =
  4   IF ." (;CODE) " CR ." CODE AT"
  5   SPACE DROP 2+ 1 1
  6   ELSE
  7   DUP ' ;S CFA =
  8    IF ." ; "
  9    ELSE 2+ NFA ID. 0 THEN
 10 THEN THEN
 11 UNTIL DROP ;
 12
 13
 14 -->
 15
```

```
SCR # 129
  0 ( DECOMPILER END                         )
  1 : Q 0 IFS ! CR GETP ." : "
  2   DUP NFA ID. 2 SPACES
  3   CFA DUP @ ' GETP CFA @ =
  4   IF LP ELSE
  5
  6   DUP @ 3775 =
  7    IF ." CONSTANT " 2+ ? 0 THEN
  8   DUP @ 3803 =
  9    IF ." VARIABLE " 2+ ? 0 THEN
 10   DUP ' DOES> CFA =
 11    IF ." DOES> ( PFA ) " @ 2+ LP
 12     0 THEN
 13     -DUP IF ." CODE AT " ? THEN
 14  THEN CR ;
 15
```

6.2 Decompiler

Im Programm wird durch 15 ARRAY IFT ein Feld für 15 Zahlen angelegt. Mit N N' IFT ! wird die Zahl N in das Element mit dem Index N' gespeichert. Von dort kann es mit N IFT @ wieder gelesen werden. Im Programm wird dieses Feld dazu benutzt, die Rücksprungadressen bei Schleifen zu speichern, um diese später wieder auflösen zu können. Die Variable IFS enthält die Anzahl der in IFT gespeicherten Zahlen.

Das Wort LP untersucht die Sonderfälle und druckt deren Ergebnis oder die Namensfeldadresse des gespeicherten Wortes aus. Bei einigen FORTH-Dialekten können einige Worte wie LIT, VARIABLE, CONSTANT und DOES> anders definiert sein. Für LIT erhält man den Namen durch

```
: TEST 25 ;
' TEST @ 2+ NFA ID.
```

Die Codefeldadresse von VARIABLE und CONSTANT erhält man

durch

```
0 VARIABLE TESTA
0 CONSTANT TESTB

' TESTA CFA @ .
' TESTB CFA @ .
```

Worte, die als IMMEDIATE erklärt sind, kann das Programm nicht erkennen. Dazu gehört auch das Wort BEGIN. Dafür wird nach UNTIL und REPEAT das auf das zugehörige BEGIN folgende Wort in Klammern ausgedruckt. Wird ein mit CODE in Maschinensprache definiertes Wort gefunden, so wird die Adresse, ab welcher dieser Code gespeichert ist, ausgegeben.
Die Abbildung 6.3 zeigt die Decompilierung von Q selbst.

```
: Q   0 IFS ! CR GETP ." : " DUP NFA ID.
  2 SPACES CFA DUP @ ' GETP CFA @ = IF LP
   ELSE DUP @ 3775 =
   IF ." CONSTANT " 2+ ? 0
    THEN DUP @ 3803 =
   IF ." VARIABLE " 2+ ? 0
    THEN DUP ' DOES> CFA =
   IF ." DOES> [ PFA ] " @ 2+ LP 0
    THEN -DUP
   IF ." CODE AT " ?
    THEN
   THEN CR ;
```

6.3 Decompilierung von Q

6.3 Suchen von Worten in Textfeldern

Manche FORTH-Dialekte, wie zum Beispiel poly-FORTH von FORTH-INC enthalten das Wort LOCATE ⟨NAME⟩. Damit kann man, wenn das Wort in das Wörterbuch kompiliert ist, das Textfeld suchen, in welchem es definiert wurde. Verfügt man nicht über ein solches Wort, so ist es manchmal schwierig, das Textfeld zu finden, in dem es auftritt. Das

Programm in Abbildung 6.4 ist die stark vereinfachte Version eines Programmes aus FORTH-DIMENSIONS V5, N3, S11.

```
SCR # 110
  0 ( FAW DIRECTORY SEARCH  20.3.EF)
  1 : TSK ;
  2 : ITEM 13 WORD HERE COUNT PAD
  3   2DUP C! 1+ SWAP CMOVE ;
  4 : COMPARE ( AA'N-F) ( F 0<,0,>0)
  5   OVER + SWAP DO COUNT I C@ -
  6   -DUP IF SWAP 0= LEAVE THEN
  7   LOOP IF 0 THEN ;
  8 : LOOK 0 BEGIN DROP BL WORD HERE
  9   PAD DUP C@ COMPARE DUP 0= IN @
 10   255 > OR UNTIL ;
 11 : =ITEM ( N-F) BLK @ IN @ >R >R
 12   B/SCR * BLK ! C/L IN ! LOOK
 13   DUP IF DROP 1 BLK +! 0 IN !
 14   LOOK THEN R> R> IN ! BLK ! ;
 15 -->

SCR # 111
  0 ( FAW DIRECTORY CNTD    20.3.EF)
  1   0 VARIABLE CONT 2 ALLOT
  2 : BEREICH ( NN') <BUILDS , ,
  3   DOES> 2@ CONT 2! ;
  4
  5 : SUCHE 0 ITEM CONT 2@ 1+ SWAP
  6   DO I =ITEM 0= IF DROP I LEAVE
  7   THEN LOOP DUP IF LIST ELSE
  8   ." NICHT GEFUNDEN" DROP THEN ;
  9
 10
 11
 12
 13
 14
 15 ;S
```

6.4 Suchen nach Worten in Textfeldern

Das Wort BEREICH ist ein Definitionswort, mit welchem man Worte definieren kann, die den Bereich festlegen, in welchem gesucht werden soll. Mit

110 111 BEREICH TEST

wird dem Wort TEST der Bereich von Textfeld 110 bis 111 dem Wort TEST zugeordnet. Durch den Aufruf von TEST werden die beiden Zahlen in der doppelt langen Variable CONT gespeichert. Mit

SUCHE ⟨NAME⟩

wird nun in diesem Bereich nach ⟨NAME⟩ gesucht. Wenn das Wort gefunden wird, so wird das Textfeld auf dem Bildschirm angezeigt.

Wenn es nicht gefunden wird, so erscheint die Fehlermeldung NICHT GEFUNDEN. Bei der für dieses Beispiel verwendeten FORTH-Version besteht ein Textfeld aus zwei Blöcken. Das Wort LOOK sucht in einem Block nach dem Wort. Das Wort =ITEM erweitert nun die Suche auf zwei Blöcke. Dabei wird die erste Zeile im ersten Block (Zeile 0 des Textfeldes, meistens Kommentarzeile) übersprungen. Soll dies nicht geschehen, so ist in diesem Wort C/L durch 0 zu ersetzen. Für den Vergleich der Zeichenketten wird das Wort COMPARE benutzt.

Das Wort ITEM verschiebt das darauf folgende Wort nach PAD und speichert im ersten BYTE von PAD die Länge des Wortes.

Dieses Suchen nach einem Eintrag kann auch leicht für andere Daten verwendet werden. In Abbildung 6.5 ist mit dem Editor von FORTH ein Telefonverzeichnis eingegeben worden. Aus diesem kann nun mit den in Abbildung 6.4 definierten Worten nach Namen oder Telefonnummern gesucht werden.

```
SCR # 139
  0 ( TELEFON VERZEICHNIS                )
  1 R.WAGNER      0721/814563
  2 H.MUELLER     089/745368
  3 C.SCHMITT     08024/4011
  4 B.SCHUSTER    0711/234765
  5
  6   USW. USW.
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

6.5 Telefonverzeichnis

Eine reizvolle Erweiterung dieses Programms ist eine Stichwortkartei.

6.4 Beispiel Stichwortkartei

In einem Text soll nach verschiedenen Worten gesucht werden. Der Text ist fortlaufend eingegeben. Die Eingabe der Suchbegriffe kann in beliebiger Reihenfolge geschehen. Die für das Suchen verwendeten Worte zeigt Abbildung 6.6. Das Wort ITEMS erwartet eine Reihe von Worten, die durch Leerzeichen getrennt sind. Diese Worte werden nach PAD geschoben, wobei vor jedem Wort das Längenbyte gespeichert wird. In der Variablen BS wird mitgezählt, wieviele Worte eingegeben worden sind. Die Variable LEN wird zum Speichern der Worte in PAD gebraucht. LOOKS sucht nun in einem Block nach den eingegebenen Begriffen. Ein Wort aus dem Block wird nacheinander mit den Begriffen verglichen. Wird eine Übereinstimmung gefunden, so wird die Variable BO um eins erhöht. Wenn am Ende des Suchens in einem Block der Inhalt von BO und von BS gleich ist, so sind die gesuchten Begriffe in diesem Block enthalten.

```
SCR # 130
  0 ( FAW STICHWORTKARTEI         EF)
  1 0 VARIABLE LEN
  2 0 VARIABLE BS 0 VARIABLE BO
  3 : OPAD PAD 128 0 FILL ;
  4 : ITEMS OPAD 0 DUP LEN ! BS !
  5 BEGIN BL WORD HERE COUNT LEN @
  6 PAD + 2DUP C! 1+ SWAP DUP >R
  7 CMOVE R> DUP 1+ LEN +! 1
  8 BS +! 1 = UNTIL ;
  9
 10 -->
 11
 12
 13
```

```
SCR # 131
  0 ( FAW STICHWORTKARTEI         EF)
  1 : (LKS) 0 BEGIN DROP BL WORD
  2   HERE LEN @ PAD + DUP C@
  3   COMPARE
  4   DUP 0= IN @ 255 > OR UNTIL ;
  5 : LOOKS CR
  6  BS @ 1 - 0 DUP LEN ! BO !
  7 BEGIN (LKS) 0=
  8   IF LEN @ PAD + C@ 1+ LEN +!
  9   0 IN ! -1 BS +! BS @ 1 =
 10   1 BO +! ELSE 1 THEN
 11 UNTIL BO @ = IF ." WORTE IN "
 12   ELSE ." WORTE NICHT IN "
 13 THEN BLK @ . ."  ENTHALTEN" ;
 14
 15 -->
```

```
SCR # 132
  0 ( STICHWORTKARTEI  CNTD       EF)
  1 : =ITEMS BLK @ IN @ >R >R
  2   B/SCR * BLK ! 0 IN ! LOOKS
  3   R> R> IN ! BLK ! ;
  4
```

6.6 Suchen nach mehreren Worten

Dieses Beispiel ist nicht vollständig ausprogrammiert, da der Aufruf von LOOKS davon abhängt, ob in einem Block, einem Textfeld oder nur in einem Teil des Blockes gesucht wird. In der Adresskartei in Kapitel 9 ist ein Eintrag 128 Bytes lang. Begrenzt man LOOKS auf diese Länge, so kann man zum Beispiel durch

SUCHE MUELLER MEINERZHAGEN

nach dem Herrn Müller aus Meinerzhagen suchen. Das gleiche Ergebnis liefert die Eingabe

SUCHE MEINERZHAGEN MUELLER .

Natürlich ist dies kein schnelles Suchen. Je nach Geschwindigkeit des Diskettenlaufwerks kann das Suchen einige Zeit in Anspruch nehmen. Für große Dateien wird man COMPARE durch ein in Maschinensprache geschriebenes Wort ersetzen.

```
SCR # 137
  0 KARL HEINZ OTTO BERND FRANK
  1 FRITZ ZAUSEL ADAM
  2 E.F
  3
  4
  5 TONI VRONI
  6
       OK
ITEMS KARL ADAM  OK
137 =ITEMS
WORTE IN 274  ENTHALTEN OK

0 PR#  OK
ITEMS FRITZ ZAUSEL VRONU  OK
137 =ITEMS
WORTE NICHT IN 274  ENTHALTEN OK

137 =ITEMS
WORTE IN 274  ENTHALTEN OK
ITEMS VRONI ZAUSEL FRITZ  OK
137 =ITEMS
WORTE IN 274  ENTHALTEN OK
```

Notizen



Sinustabelle mit Turtlegrafik

Die ersten beiden Textfelder in Abbildung 7.1 enthalten eine Sinustabelle. Die angegebenen Werte sind vierstellig und geben den Bereich von 0 bis 90 Grad jeweils in Schritten von einem Grad an.

Die Worte SIN und COS ersetzen auf dem Stapel die Gradzahl durch den entsprechenden Sinus- oder Cosinus Wert.

Mit diesen Worten wird eine einfache Turtlegrafik programmiert. Die maschinenspezifischen Worte sind:

WHITE setzt die Zeichenfarbe auf weiß.

BLACK setzt die Zeichenfarbe auf schwarz.

HGR schaltet die hochauflösende Grafik ein.

PLOT zeichnet einen Punkt X, Y in der gewählten Farbe.

LINE zeichnet eine Linie vom augenblicklichen Punkt zum Punkt X, Y.

Die beiden Variablen X, Y enthalten den jeweiligen Standort der "Schildkröte". Diese beginnt in der Bildmitte mit der Blickrichtung nach rechts. TURN ändert die Laufrichtung. Das Wort DRAWTO benötigt zwei Angaben, den Winkel und die Länge auf dem Stapel. Die Einheit der Länge einer Linie ist der Abstand von zwei Bildpunkten auf dem Bildschirm.

95 40 DRAWTO ändert den Winkel um 95 Grad und zeichnet dann eine Linie von 40 Bildpunkten.

```
SCR # 75
  0 ( TTG SIN-TAB                              EF)
  1 : TABLE <BUILDS  DOES>
  2   SWAP 2 * + @ ;
  3    TABLE TSIN
  4 0000 , 0175 , 0349 , 0523 ,
  5 0698 , 0872 , 1045 , 1219 ,
  6 1392 , 1564 , 1763 , 1908 ,
  7 2079 , 2250 , 2419 , 2588 ,
  8 2756 , 2924 , 3090 , 3256 ,
  9 3420 , 3584 , 3746 , 3907 ,
 10 4067 , 4226 , 4384 , 4540 ,
 11 4695 , 4848 , 5000 , 5150 ,
 12 5299 , 5446 , 5592 , 5736 ,
 13 5878 , 6018 , 6157 , 6293 ,
 14 6428 , 6561 , 6691 , 6820 ,
 15           -->
```

```
SCR # 76
  0 ( TTG SIN-TAB CNTD                         EF)
  1 6947 , 7071 , 7193 , 7314 ,
  2 7431 , 7547 , 7660 , 7771 ,
  3 7880 , 7986 , 8090 , 8192 ,
  4 8290 , 8387 , 8480 , 8572 ,
  5 8660 , 8746 , 8829 , 8910 ,
  6 8988 , 9063 , 9135 , 9205 ,
  7 9272 , 9336 , 9397 , 9455 ,
  8 9511 , 9563 , 9613 , 9659 ,
  9 9703 , 9744 , 9781 , 9816 ,
 10 9848 , 9877 , 9903 , 9925 ,
 11 9945 , 9962 , 9976 , 9986 ,
 12 9994 , 9998 , 10000 ,
 13
 14 ( END SIN-TAB ) -->
 15
```

```
SCR # 77
  0 ( TTG SIN COS                    EF)
  1 : (SIN) ( N-N') DUP 90 >
  2   IF 180 SWAP - THEN TSIN  ;
  3
  4 : SIN ( N-N') 360 MOD DUP 0<
  5   IF 360 + THEN DUP 180 >
  6   IF 180 - (SIN) MINUS ELSE
  7   (SIN) THEN ;
  8
  9 : COS ( N-N') 360 MOD 90 + SIN ;
 10
 11
 12
 13
 14    -->
 15

SCR # 78
  0 ( TTG COMMANDS                   EF)
  1   140 VARIABLE X  95 VARIABLE Y
  2 : @XY X @ Y @ ;
  3 : SI ( N'N-N") SWAP SIN 10000
  4   */ ;
  5 : CO ( N'N-N") SWAP COS 10000
  6   */ ;
  7 : TURTLE ( -N') 140 X ! 95 Y !
  8   HGR BLACK @XY PLOT 0 ;
  9 : TURN ( N'-N") +  ;
 10
 11 : DRAW ( N'N-N') OVER SWAP
 12   DUP >R CO X +! DUP R>
 13   SI MINUS Y +! @XY LINE ;
 14 : DRAWTO ( N'N) >R TURN R>
 15   DRAW ; -->

SCR # 79
  0 ( TTG COMMANDS CNTD              EF)
  1 : TPLOT @XY PLOT ;
  2 : GOTO ( XY) 95 SWAP - Y !
  3   140 SWAP + X ! TPLOT ;
  4
  5     -->
```

```
SCR # 80
  0 ( TTG                          EF)
  1 : SQUARE 4 0 DO 90 40 DRAWTO
  2   LOOP  ;
  3 : SQS 36 0 DO SQUARE 100 TURN
  4   40 DRAW LOOP ;
  5 : SQ 0 DO 95 40 DRAWTO LOOP ;
  6 : SQ1 36 0 DO SQUARE 10 TURN
  7   LOOP ;
  8
  9        —>
 10
 11
```

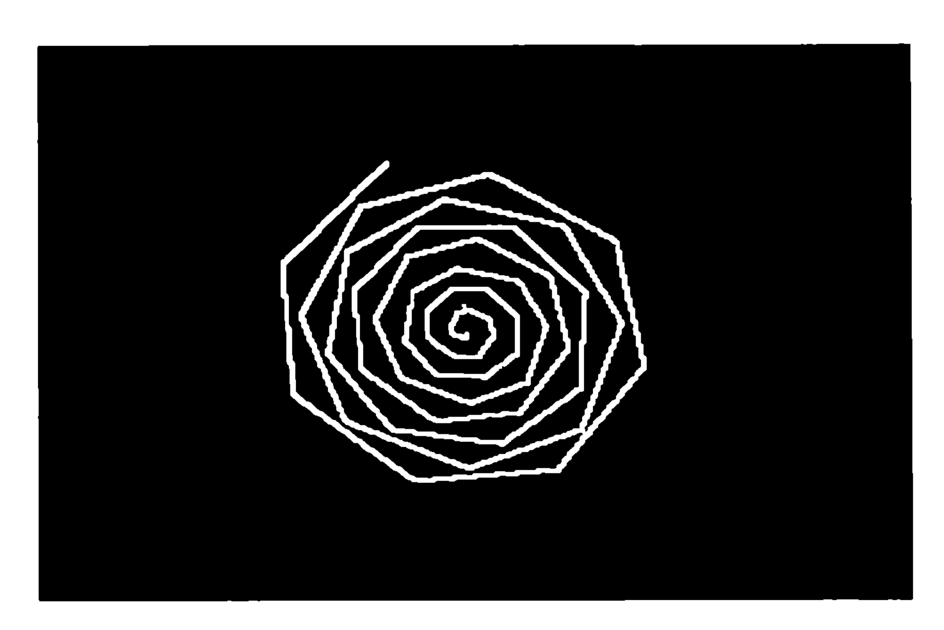
```
SCR # 81
  0 ( TTG                          EF)
  1 : TI TURTLE WHITE TPLOT ;
  2 : SQ2 60 0 DO 90 TURN 60 I -
  3   DRAW LOOP DROP ;
  4 : SQ3 42 0 DO 92 TURN 60 I -
  5   DRAW LOOP DROP ;
  6 : SQ4 30 -60 GOTO 8 0 DO
  7   45 TURN 60 DRAW LOOP DROP ;
  8 : SQ5 30 -60 GOTO 60 0 DO
  9   47 TURN 60 I - DRAW LOOP
 10   DROP ;
 11
 12
```

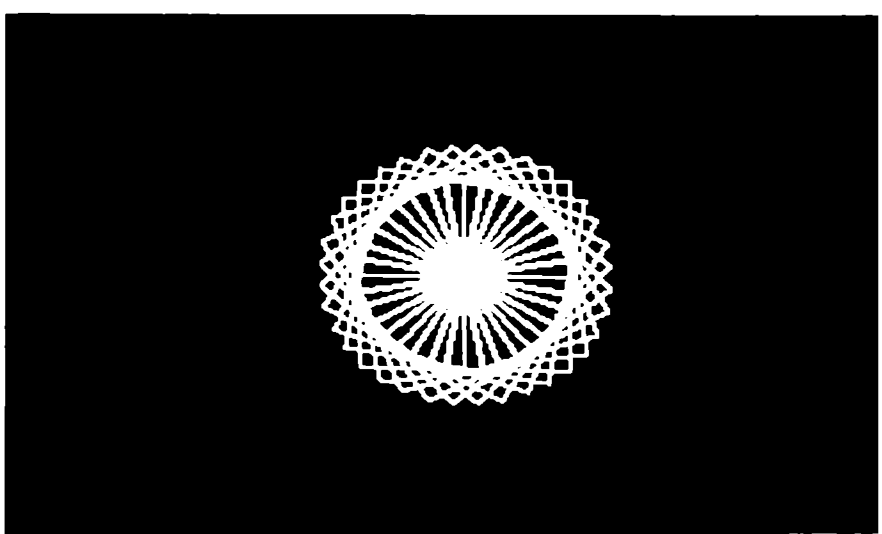
```
SCR # 138
  0 ( SINUS ANNAEHERUNG                  )
  1 : 3PICK >R OVER R> SWAP ;
  2
  3 : SIN ( MCS-MC'S')
  4   3PICK 3PICK SWAP / + SWAP
  5   3PICK 3PICK SWAP / - SWAP ;
  6
  7
```

7.1 Sinustabelle und Turtlegrafik

In Textfeld 80 und 81 sind einige Beispiele angegeben.

TI initialisiert die Grafik. Die Beispiele SQ2 und SQ4 sind in Abbildung 7.2 gezeigt.





7.2 Turtlegrafik

Abbildung 7.3 zeigt eine andere Art, eine Sinusfunktion auszugeben. Von einem Ausgangswert für den Cosinus und den Sinus wird der nächste Wert nach folgender Formel berechnet:

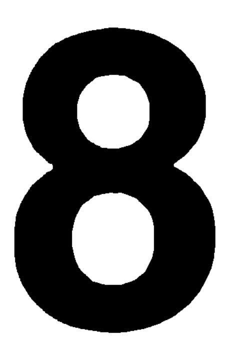
SIN (N+1) = 1/M * COS (N)

COS (N+1) = 1/M * SIN (N) .

```
SCR # 138
  0 ( SINUS ANNAEHERUNG                    )
  1 : 3PICK >R OVER R> SWAP ;
  2
  3 : SIN ( MCS-MC'S')
  4   3PICK 3PICK SWAP / + SWAP
  5   3PICK 3PICK SWAP / - SWAP ;
  6
  7


      50 10000 0 SIN  OK
      .S
      50 9996 200  OK
      SIN  OK
      .S
      50 9989 399  OK
      SIN  OK
      .S
      50 9978 598  OK
      SIN  OK
      .S
      50 9963 797  OK
```

7.3 Näherungsweise Berechnung von Sinus und Cosinus



Rekursion

In FORTH ist die Programmierung von rekursiven Verfahren, wie z. B. in PASCAL von vornherein nicht möglich.

Wie dies dennoch möglich ist, soll am Beispiel der "Türme von Hanoi" gezeigt werden. Dieses Beispiel wurde von Herrn Dr. E. D. Schmitter programmiert.

8.1 Die Türme von Hanoi

Im folgenden wird die Programmierung eines rekursiven Verfahrens einmal ausführlich dargestellt.

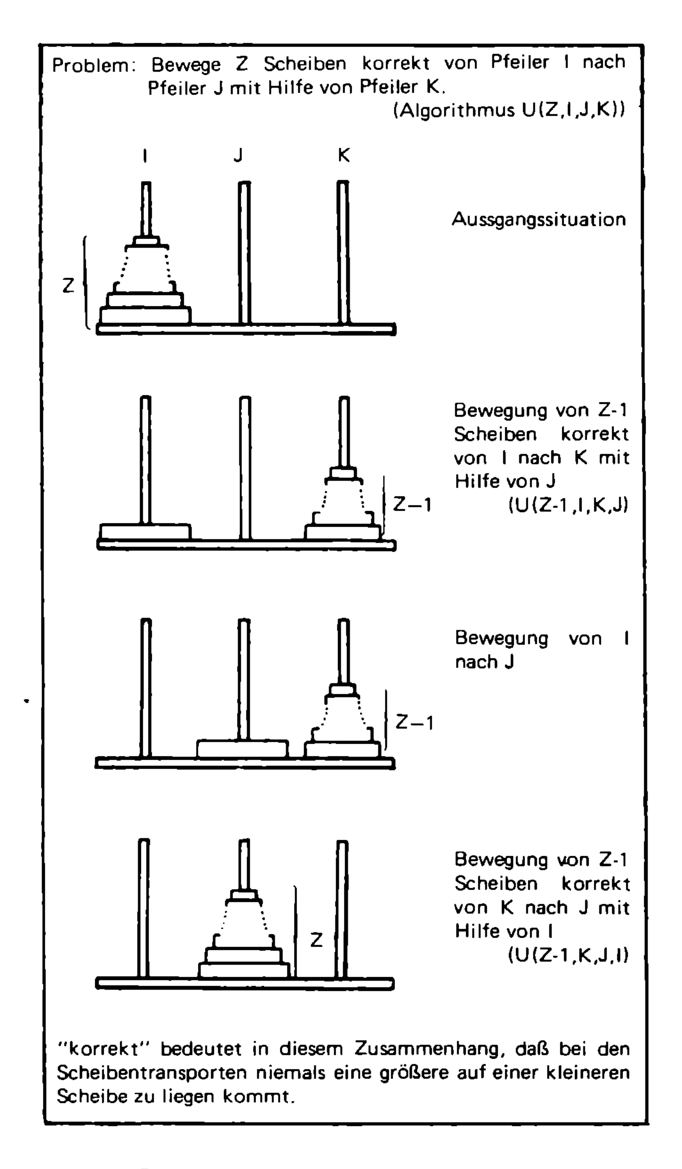
Betrachten wir zur Illustration das Hanoi-Problem mit 3 Scheiben auf Pfeiler 1. Die Umschichtung auf Pfeiler 2 erfolgt am kürzesten mit folgenden Schritten:

Bewege oberste Scheibe von Pfeiler 1 nach 2
1 nach 3
2 nach 3
1 nach 2
3 nach 1
3 nach 2
1 nach 2

Am besten macht man sich die Vorgänge mit Hilfe von 3 verschieden großen Papierschnitzeln klar.

Bei genauer Betrachtung der angegebenen Umschichtsequenz ergibt sich

folgendes Vorgehen: In den ersten 3 Schritten werden die obersten beiden Scheiben von 1 nach 3 transportiert. Danach wird die nun freiliegende unterste Scheibe von 1 nach 2 bewegt. In den letzten 3 Schritten werden die beiden Scheiben, die auf 3 liegen, nach 2 bewegt, womit der Transport beendet ist.



8.1 Prinzip des "Türme von Hanoi"-Algorithmus

Für Z Scheiben läßt sich das folgende, allgemeine Prinzip angeben (siehe auch Abbildung 8.1):

1. Bewege Z-1 Scheiben von 1 nach 3.
2. Bewege die letzte, unterste Scheibe von 1 nach 2.
3. Bewege die Z-1 Scheiben von 3 nach 2.

Damit ist der Turm von 1 nach 2 transportiert. Man löst das Problem, Z Scheiben zu transportieren dadurch, daß man 2 mal Z-1 Scheiben transportiert. Der Transport von Z-1 Scheiben wird seinerseits auf den von Z-2 Scheiben zurückgeführt – und so fort . . .

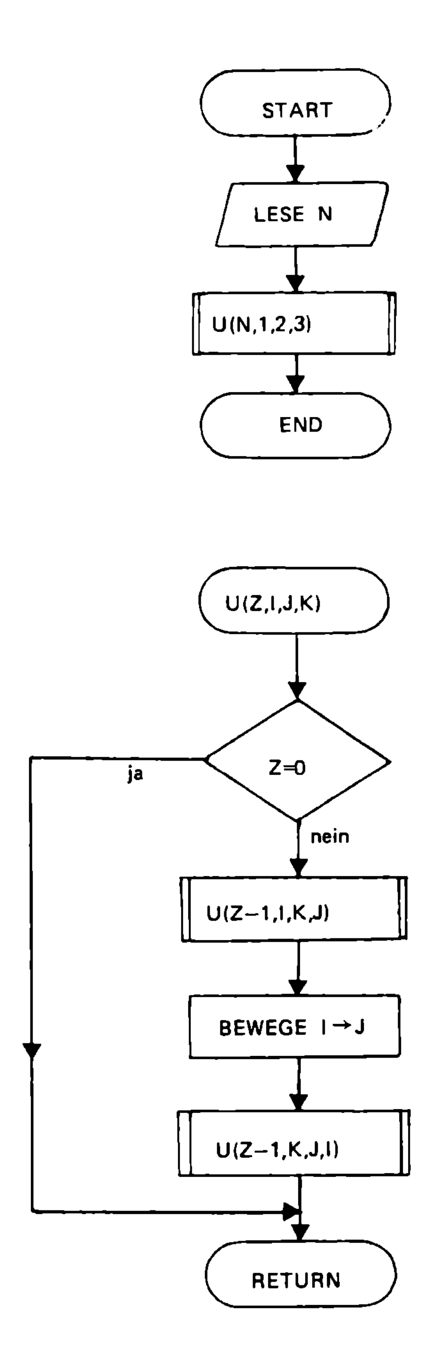
Wir wollen diesen Algorithmus, der regelmäßig Z Scheiben von Pfeiler I nach Pfeiler J mit Hilfe von Pfeiler K transportiert, U(Z,I,J,K) nennen. Er ist rekursiv, weil er sich in der Form U(Z–1,I,K,J) bzw. U(Z–1,K,J,I) selbst aufruft (siehe dazu Bild 8.1).

Damit die Sache durchsichtig wird, bringen wir den Algorithmus zunächst in die Form eines Flußdiagramms (Abbildung 8.2). Der Vorgang startet mit der Vorgabe der Scheibenzahl N und ruft dann den Algorithmus U(N,1,2,3) auf, um N Scheiben von 1 nach 2 mit Hilfe von 3 zu bringen. Wieso dieses Flußdiagramm das Problem wirklich löst, bleibt – wie oft bei rekursiven Algorithmen – zunächst unklar. Um zu sehen, was wirklich passiert, nehmen wir wieder unseren 3-Scheiben-Turm und verwenden den Algorithmus gemäß Flußdiagramm darauf an (siehe hierzu Abbildung 8.3).

Rekursion

Der Algorithmus startet mit dem Aufruf U(3,1,2,3). D. h. in U(Z,I,J,K) erfolgt die Belegung Z:=3;I:=1; J:=2;K:=3. Wegen Z > 0 wird U(Z–1,I, K,J) (siehe Flußdiagramm, Abbildung 8.2) aufgerufen, also U(2,1,3,2).

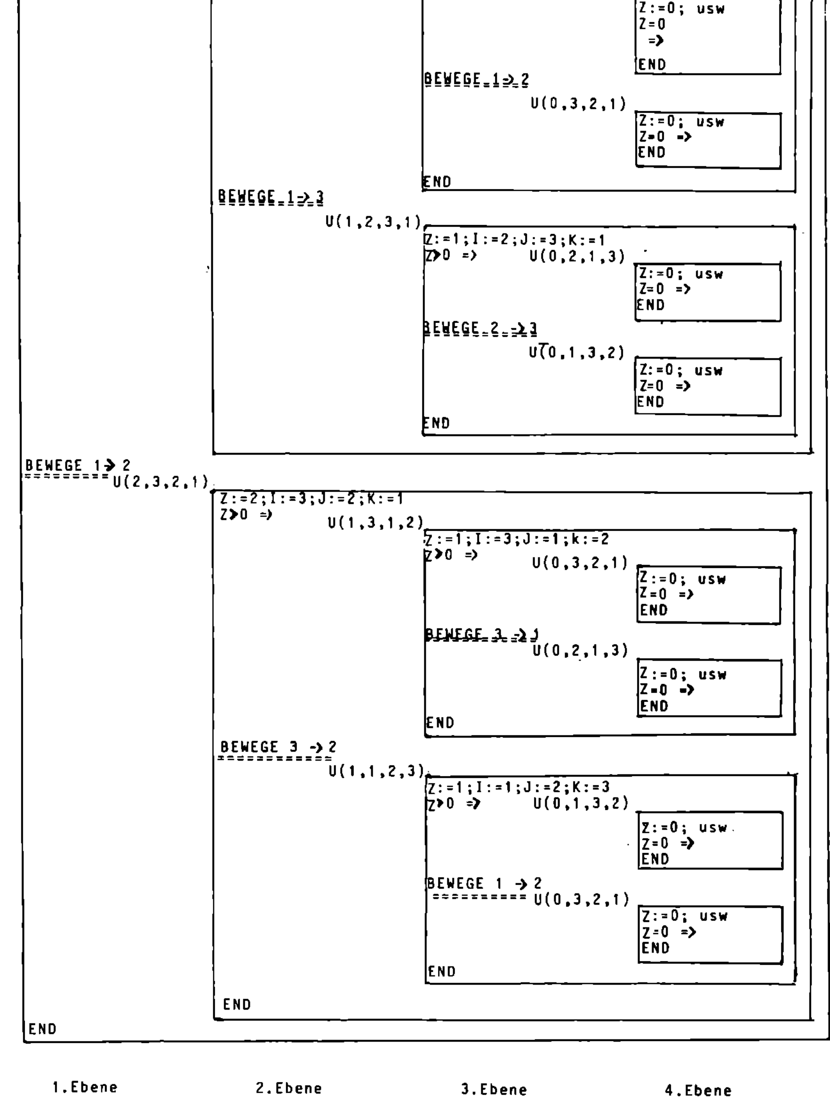
Wir treten damit in die nächste (2.) Ebene der Rekursion ein. In dieser Ebene werden neue Variable Z,I,J,K definiert und belegt: Z:=2;I:=1; J:=3;K:=2, ohne daß die alten Werte der Ebene 1 verloren gehen.



8.2 Flußdiagramm des "Türme von Hanoi"-Algorithmus

HANOI mit Z:=3 Scheiben

U(3,1,2,3)



8.3 Die Rekursionsebenen des Hanoi-Problems mit 3 Scheiben

Genau genommen müßte man, um ganz deutlich zu machen, daß hier neue Variable definiert werden, diese mit der Nummer der Ebene indizieren, also schreiben Z(2):=2;I(2):=1;J(2).=3;K(2):=2. Die Variablen der ersten Ebene müßte man entsprechend mit (1) versehen. In der Abbildung 8.3 ist die Indizierung weggelassen, da die Ebenenzuordnung aus dem Diagramm hervorgeht.

Es geht weiter: Da Z(2) 〉 0 ist, wird U(Z(2)−1,I(2),K(2),J(2))= U(1,1,2,3) aufgerufen. Damit treten wir in die 3. Ebene ein, wo die Belegung Z(3):=1;I(3):=1;J(3):=2;K(3):=3 erfolgt. Da Z(3) 〉 0 ist, wird mit dem Aufruf U(Z(3)−1,I(3),K(3),J(3))=U(0,1,3,2) in die 4. Ebene eingetreten. Da Z(4)=0 ist, erfolgt auf dieser Ebene ein sofortiger Sprung zum Ende des Algorithmus (auf dieser Ebene aber nur !).

Gemäß Flußdiagramm, Abbildung 8.2, wird das Ende durch einen RETURN-Befehl gebildet, der einen Rücksprung in die vorhergehende Ebene, hier Ebene 3 bewirkt. Hier geht es mit der Anweisung nach dem Aufruf U(Z−1,I,K,J) weiter, da letzterer auf der Ebene 3 zunächst erledigt ist. Die nächste Anweisung lautet gemäß Flußdiagramm: BEWEGE I(3) → J(3), also BEWEGE 1 → 2.

Danach kommt der Aufruf U(Z(3)−1,K(3),J(3),I(3))=U(),3,2,1). Das bedeutet Wiedereintritt in die 4. Ebene. Da Z(4)=0 ist, erfolgt sofortiger Rücksprung in die Ebene 3. Weitergemacht wird auf dieser Ebene mit der Anweisung nach dem U(Z−1,K,J,I) Aufruf. Die nächste Anweisung ist aber der RETURN-Befehl, der nun mehr einen Rücksprung in die Ebene 2 bewirkt. Auf Ebene 2 ist der letzte abgearbeitete Befehl der Aufruf U(Z(2)−1,I(2),K(2),J(2)). Es folgt die Anweisung BEWEGE I(2) → J(2), also BEWEGE 1 → 3.

Anschließend treten wir mit dem Aufruf U(Z(2)−1,K(2),J(2),I(2))= U(1,2,3,1) wieder in die 3. Ebene ein.

Wir wollen die Detaildiskussion an dieser Stelle abbrechen, da man sich anhand von Abbildung 8.3 alle weiteren Einzelheiten in der beschriebenen Manier überlegen kann.

Wir fassen zusammen: Ruft sich der Algorithmus selbst auf, so bedeutet dies den Eintritt in eine neue Ebene. Alle im Algorithmus vorkommen-

den Variablen müßten genau genommen mit dieser Ebene ihnen neue Werte zugeordnet werden, ohne daß die der vorangehenden Ebenen verlorengehen. In verschiedenen Programmiersprachen wird das Problem durch die Einführung sog. "lokaler" Variabler gelöst, bei denen die Wertzuweisung in einem Unterprogramm nur innerhalb der gerade beschriebenen Rekursionsebene gültig ist. In PASCAL z. B. sind in der PROCEDURE U(Z,I,J,K) die Variablen Z,I,J,K als lokal zu betrachten. BASIC hingegen kennt das Konzept der lokalen Variablen nicht. Hier sind alle Variablen "global", was für uns bedeutet, daß die Variablen als indizierte Variable Z(L),I(L),J(L) und K(L) definiert werden müssen. Der Index L bezieht sich auf die Rekursionsebene, in welcher der Algorithmus gerade arbeitet. Wir kommen auf diesen Punkt zurück.

RETURN schließlich bedeutet Rücksprung in die vorangegangene Rekursionsebene und Fortsetzung des Algorithmus auf dieser Ebene mit der Anweisung nach dem letzten Selbstaufruf.

15 Milliarden Jahre

Als nächstes interessiert uns die Frage, wieviele Rekursionsebenen beim N-Scheiben-Hanoi Problem auftreten.

Laut Flußdiagramm wird durch fortgesetzte Aufrufe von U(Z-1, . . .) in N Schritten 0 erreicht, wobei jedesmal eine neue Ebene betreten wird. Der Aufruf U(0,. . .) hat den Eintritt in die Ebene N+1 zur Folge, von wo der sofortige Rücksprung in die N-te Ebene erfolgt. Somit hat das N-Scheiben-Problem N+1 Rekursionsebenen.

In jeder Ebene gibt es laut Flußdiagramm genau eine BEWEGE Anweisung, bis auf die Ebene N+1, die wegen Z=0 mit dem sofortigen Rücksprung nach Ebene N keine BEWEGE Anweisung liefert. Wie oft wird nun jede Ebene aufgerufen ?

Betrachten wir Abbildung 8.3, so sieht man, daß

Ebene 1   1 mal,
Ebene 2   2 mal,
Ebene 3   4 mal,
Ebene 4   8 mal

aufgerufen wird. Man überlegt mit nicht allzu großer Mühe, daß allgemein die k-te Ebene 2^{k-1} mal aufgerufen wird (jede folgende Ebene wird wegen des 2-maligen U-Aufrufs im Algorithmus doppelt so oft aufgerufen, wie die vorangehende).

Die Gesamtzahl aller Ebenenaufrufe ohne Ebene N+1 ist also

$$1+2+4+8+\ldots = \sum_{k=1}^{N} 2^{k-1} = 2^{N}-1$$

Dies ist auch die Gesamtzahl der Bewegungen, die mit diesem Algorithmus nötig sind, um das N-Scheiben-Problem zu lösen. Dies sind

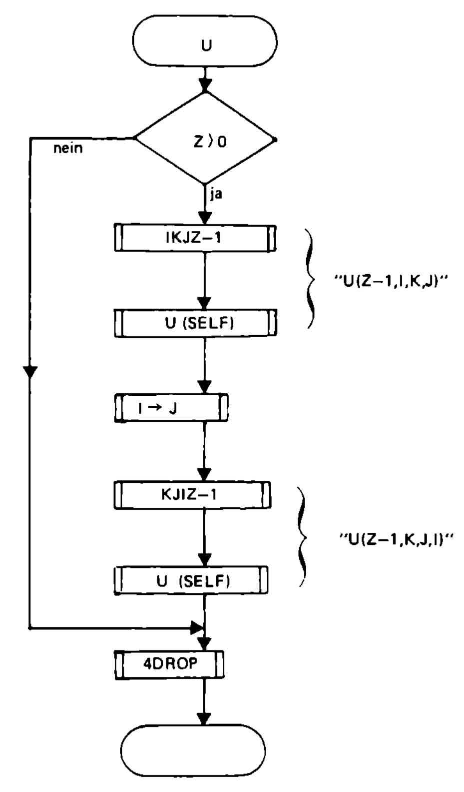
bei 3 Scheiben 7 Anweisungen,
bei 10 Scheiben 1023 Anweisungen und
bei 64 Scheiben $2^{63}-1$ = 9223372036854775807 = ca. 10^{19} Anweisungen.

Selbst wenn also beim 64-Scheiben-Problem, dem sich die Weisen von Hanoi widmen, alle 1/20 Sek. eine BEWEGE Anweisung käme, die von den Weisen sofort ausgeführt würde (wie immer diese das so schnell realisieren), so bräuchten sie doch etwa 15 Milliarden Jahre, um den Turm umzuschichten. Übrigens schätzt man das Alter unserer Erde auf 4,7 Milliarden Jahre und das des Weltalls auf 10 bis 20 Milliarden Jahre . . .

8.2 Das FORTH-Programm

Abbildung 8.4 zeigt das Flußdiagramm des Türme-von-Hanoi-Algorithmus, zugeschnitten auf Verarbeitung mit FORTH. Jedem Kästchen entspricht ein FORTH-Wort — woraus die Vorteile dieser Sprache bezüglich eines strukturierten Programmierstils sofort deutlich werden.

Abbildung 8.5 enthält das FORTH-Programm, sowie eine Beschreibung der auftretenden FORTH-Worte in Hinsicht auf Ihre Wirkung auf den Parameterstapel. Einer besonderen Erklärung bedarf an dieser Stelle die Technik der Rekursion (Selbstaufruf eines FORTH-Wortes).



8.4 Flußdiagramm für das FORTH-Wort U (I J K Z →)

```
SCR # 114
  0 ( FAW REKURSION                    DRS  )
  1 ( TUERME VON HANOI  )
  2
  3 : SELBST LATEST PFA CFA
  4   [COMPILE] LITERAL ; IMMEDIATE
  5 : SELF [COMPILE] SELBST
  6   EXECUTE ;
  7
  8 : 4DUP 4 0 DO SP@ 6 + @ LOOP ;
  9 : IKJZ-1 4DUP 1 - ROT ROT SWAP
 10   ROT ;
 11 : I->J 4DUP 2DROP SWAP CR .
 12   ." -> " . ;
 13 : KJIZ-1 4DUP 1 - SP@ 2 + @
 14  SP@ 8 + @ SP@ 6 + ! SP@ 8 + ! ;
 15    -->
```

```
SCR # 115
  0 ( FAW TUERME V. HANOI CNTD DRS )
  1 : 4DROP 2DROP 2DROP ;
  2
  3 : U DUP 0 > IF IKJZ-1 SELF I->J
  4   KJIZ-1 SELF ENDIF 4DROP ;
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15 ;S
 OK
```

Ausführung N=4

```
1 2 3 4 U
1 -> 3
1 -> 2
3 -> 2
1 -> 3
2 -> 1
2 -> 3
1 -> 3
1 -> 2
3 -> 2
3 -> 1
2 -> 1
3 -> 2
1 -> 3
1 -> 2
3 -> 2  OK
```

Beschreibung der FORTH-Worte:

SELF – unter Verwendung von SELBST – bewerkstelligt allgemein den Selbstaufruf (Rekursion) des FORTH-Wortes, in dessen Definition es auftritt. Näheres im Text.

4DUP	(I J K Z → I J K Z I J K Z)
IKJZ–1	(I J K Z → I K J Z–1)
I → J	(I J K Z → I J K Z) Ausgabe von "I → J" auf Bildschirm
KJIZ–1	(I J K Z → K J I Z–1)
4DROP	(I J K Z →)
U	(I J K Z →) Türme von Hanoi Algorithmus "U(Z,I,J,K)"

8.5 Das FORTH-Programm

Ein FORTH-Wort kann sich nicht ohne weiteres selbst aufrufen:

Eine Definition der Art

```
: WORT . . . . WORT . . . . ;
```

führt zunächst zur Fehlermeldung

```
WORT?
```

Man kann FORTH die Rekursion aber beibringen. Dies geschieht beispielsweise mit den beiden Worten

```
: SELBST LATEST PFA CFA (COMPILE) LITERAL;
  IMMEDIATE
: SELF (COMPILE) SELBST EXECUTE ;
```

deren Bedeutung wir jetzt erläutern.

Zunächst zu SELBST:
LATEST holt die Namensfeldadresse des zuletzt definierten Wortes auf den Stapel. PFA CFA wandelt diese über die Parameterfeldadresse in die zugehörige Codefeldadresse um.

LATEST PFA CFA innerhalb einer Wortdefinition: WORT . . . . ; benutzt – holt also die Codefeldadresse von WORT auf den Stapel, wobei dies aber schon während der Compilation von WORT – nicht erst während der Ausführung – erfolgen soll. Zu diesem Zweck ist SELBST als IMMEDIATE deklariert.

LITERAL ist ein Wort mit IMMEDIATE-Charakter. Während der Compilation nimmt es die auf dem Stapel liegende letzte Zahl (TOS) – hier also die Codefeldadresse von WORT in die Definition von WORT auf, um sie bei der Ausführung (RUNTIME) von WORT wieder auf dem Stapel abzulegen.

Damit nun aber – wegen des IMMEDIATE-Charakters von LITERAL – LITERAL nicht schon während der Compilation von SELBST ausgeführt wird, schreiben wir (COMPILE) davor. Dadurch wird der IMMEDIATE-Charakter von LITERAL an dieser Stelle (lokal) auf-

gehoben und LITERAL wird in die Definition von SELBST compiliert.

Zusammen: SELBST compiliert in die Definiton von WORT (wenn es in ihr vorkommt) die Codefeldadresse von WORT hinein – und legt sie bei der Ausführung von WORT auf dem Stapel ab.

EXECUTE interpretiert die gerade oben auf dem Stapel befindliche Zahl als Codefeldadresse eines Wortes und startet die Ausführung dieses Wortes.

SELBST EXECUTE startet damit also beim Lauf (RUNTIME) von WORT das FORTH-Wort WORT wieder neu an.

Diese beiden Worte fassen wir nun im neuen Wort SELF zusammen (s. o.). Da nun SELBST IMMEDIATE-Charakter hat, muß durch Vorschalten von (COMPILE) die Sofortausführung von SELBST bei der Compilation von SELF verhindert werden: SELBST wird so in die Definition von SELF "hineincompiliert" und tritt erst bei der Ausführung von SELBST in Aktion.

```
: WORT . . . . SELF . . . . ;
```

ist jetzt rekursiv: SELF hat den Selbstaufruf von WORT zur Folge.

Es existieren FORTH-Versionen, die ein Wort mit der Bedeutung von SELF (oft auch MYSELF genannt) bereits im Wörterbuch haben. Die hier verwendete Fig. FORTH-Version verfügt nicht von vorneherein darüber.

Doch weiter zur Programmbesprechung:
Die Worte 4DUP, 4DROP, IKJZ–1, KJIZ–1 und I → J manipulieren einen Satz von 4 Zahlen I J K Z auf dem Stapel in der in Abbildung 8.5 beschriebenen Weise.

Das Wort U enthält schließlich den Türme-von-Hanoi-Algorithmus gemäß Flußdiagramm (Abbildung 8.4).

U erwartet auf dem Stapel zu Beginn den Parametersatz 1 2 3 N, wobei

N die Scheibenzahl angibt. Ein Ausführungsbeispiel befindet sich in Abbildung 8.5.

Verfügt das FORTH-Wörterbuch über das Wort SMUDGE (z. B. Fig-FORTH), so läßt sich das Rekursionsproblem noch anders lösen (siehe Zeitschrift mc 9/83, S. 6, Leserbrief "FORTH"): Dazu muß man wissen, daß jedes FORTH-Wort im Namensfeld ein "Suchbit" mit sich führt. Ist das Suchbit = 1, so wird das Wort bei der Suche im Wörterbuch übergangen, ist es = 0, so wird es aufgefunden. Nach Beginn einer Wortdefinition wird das Suchbit erst einmal auf 1 gesetzt, was insbesondere bedeutet, daß es sich nicht selbst aufrufen kann. Erst, wenn kein Fehler bei der Compilation auftritt und das Endzeichen " ; " abgearbeitet ist, wird das Suchbit auf 0 gesetzt und das Wort freigegeben. SMUDGE invertiert das Suchbit. Wir überlisten den Rechner also folgendermaßen:

```
: U [ SMUDGE ] . . . U | . . . U . . . [ SMUDGE ] ;
```

Nach Compilation von : U ist das Suchbit 1, wird dann auf 0 gesetzt durch SMUDGE, so daß der Selbstaufruf U auch findet. Am Ende muß dieser Vorgang umgekehrt werden, um den Normalzustand wieder herzustellen, bevor " ; " compiliert wird. Damit SMUDGE nicht mit der Definition compiliert wird, sondern bei der Compilation von U direkt ausgeführt wird, erscheinen die eckigen Klammern, die den Compiler ab- und einschalten.

```
SCR # 112
  0 ( FAW REKURSION                  DRS  )
  1 ( TUERME VON HANOI  )
  2 ( ZWEITE VERSION  )
  3
  4 : 4DUP 4 0 DO SP@ 6 + @ LOOP ;
  5 : IKJZ-1 4DUP 1 - ROT ROT SWAP
  6   ROT ;
  7 : I->J 4DUP 2DROP SWAP CR .
  8    ." -> " . ;
  9 : KJIZ-1 4DUP 1 - SP@ 2 + @
 10  SP@ 8 + @ SP@ 6 + ! SP@ 8 + ! ;
 11    -->
 12
```

```
SCR # 113
  0 [ FAW TUERME V. HANOI CNTD DRS ]
  1 : 4DROP 2DROP 2DROP ;
  2
  3 : U [ SMUDGE ]
  4   DUP 0 > IF IKJZ-1 U I->J
  5   KJIZ-1 U ENDIF 4DROP
  6   [ SMUDGE ] ;
  7
  8
  9
 10
 11
 12
 13
 14 ;S
 15
```

8.6 Rekursion mit SMUDGE

Abbildung 8.6 zeigt das FORTH-Programm in der "SMUDGE"-Version. Die Laufzeit wird allerdings nur unwesentlich verkürzt. Mit N=10 läuft die "SELBST"-Version 65 sek., während die SMUDGE-Version ca. 60 sek. beansprucht.

9 Adressverwaltung mit Rechnungsschreiben

Die Textfelder 150 bis 171 stellen eine Adressverwaltung dar, die auch, bis auf eine Änderung, im BUSIPACK verwendet wird. Auch die Lagerverwaltung ist ähnlich aufgebaut. Diese Adressverwaltung wird durch ein Programm zum Schreiben einer Rechnung ergänzt. Im Gegensatz zum BUSIPACK wird hier nur ein Artikel, zum Beispiel eine Zeitschrift, die an verschiedene Kunden geschickt wird, verwaltet.

Die Textfelder 140 bis 144 enthalten die schon teilweise besprochenen Worte zum Ausdrucken und Formatieren von Text. Textfeld 143 übernimmt das Speichern der Adressen auf Diskette. Diese werden ab Block 80 gespeichert. Die vorderen Blöcke sind für Versuche mit binären Bäumen und verketteten Listen freigehalten. In den ersten beiden Bytes von Block 80 ist die Nummer des nächsten Eintrags gespeichert. Diese Zahl gibt die in der Datei gespeicherten, um Eins erhöht, an. Sie ist als FIRST# bezeichnet. Das Wort #INDEX berechnet aus dem Index N den Block, in welchem die gesuchte Adresse gespeichert ist. Das Wort lautet für die meisten Rechner:

```
: #INDEX (N--A) RECLEN B/BUF */MOD
  START + BLOCK + ;
```

Die im Ausdruck angegebene Version ist für den C-64 bestimmt. Dort darf Block 357 nicht beschrieben werden. In diesem Block sind Informationen für das DOS des Commodore Rechners gespeichert. Die Worte !MEM und @ MEM speichern den Eintrag N auf Diskette oder holen ihn von dort. Das Wort !ENTRY speichert einen Eintrag an die Stelle zurück, deren Nummer in der Variablen #NR gespeichert ist.

Die Worte in Textfeld 144 speichern ein Datum im Wörterbuch und auf Diskette. Dort wird es in den auf FIRST# folgenden Bytes gespeichert.

Eine Adresse wird bei der Eingabe in eine Maske eingetragen. Diese Maske zeigt Abbildung 9.1.

```
  1
 ----------------------------------
 -VERLAG DER COMPUTERFREUNDE  ------
 ----------------------------------
 -ABT. RECHNUNGSWESEN    ----------
 ----------------------------------
 -GRUENE BAUM STR. 27    ----------
 ----------------------------------
 -D-8000 -MUENCHEN            -----
 ----------------------------------
 -     -123456 -089/9876544-----  -
 ----------------------------------
  12345-1234567-12345678901-

WAS 2 PR# SCRP 0 PR#

--------------------------------------
EIN MEHR ENDE EXIT LOESCHEN MENU
NA CO ST PLZ ORT C1 C2 TEL
```

9.1 Maske für die Eingabe einer Adresse

Das Wort DESCR erzeugt den Kopf für eine doppelt lange Zahl. Diese Variablen (NA), (CO) usw. enthalten die relative Anfangsadresse zu PAD und die Länge des Eintrags. Der Eintrag für die Straße beginnt zum Beispiel bei PAD+51 und ist 23 Bytes lang.

Worte im nächsten Textfeld holen diese beiden Zahlen auf den Stapel. Die Worte CNA, CCO setzen den Cursor an die entsprechende Stelle in der Maske. CU ist maschinenabhängig. Die Definition ist

CU (ZS) setzt den Cursor in die Zeile Z und die Spalte S.

Die folgenden Worte löschen den Inhalt eines Eintrags auf dem Bildschirm und in PAD. Die Worte für die Eingabe einer Adresse sind im Textfeld 154 vereinbart. Dazu einige allgemeine Bemerkungen.

Von den Sprachen PASCAL und BASIC ist man gewöhnt, daß man in einem Programmschlauch steckt. Die Wiederholung einer Eingabe ist erst am Ende einer oft recht langen Eingabefolge möglich. In diesem Sinne ist auch die Eingabe einer Adresse im BUSIPACK programmiert. Dieses Programm zeigt eine andere Form einer Eingabe. Das Wort NA löscht den Inhalt des Namensfeldes sowohl in PAD, als auch auf dem Bildschirm, setzt den Cursor an den Anfang des Feldes und wartet auf die Eingabe von Zeichen. Nach RETURN, oder wenn das Feld voll ist, springt der Cursor wieder zu der Frage WAS. Dann kann ein anderes Wort, z. B. ORT eingegeben und dort kann ein Eintrag gemacht werden. In dieser Form sind bei WAS alle anderen Worte zugänglich. Diese Eingabe ist bei der Korrektur von Eintragungen recht nützlich, jedoch bei der Eingabe von vielen Adressen recht langsam und aufwendig. Im vorliegenden Programm können beide Eingabeformen, Einzeleingabe und fortlaufende Eingabe benutzt werden. Das Wort Ein setzt den Cursor an den Anfang des Namensfeldes. In das nächste Feld wird mit RETURN gesprungen, oder wenn das Feld voll beschrieben ist, wird es übergangen. Hier entfällt auch die Nachfrage, ob die Eintragung richtig ist oder nicht. Nach der fortlaufenden Eingabe kann dann ein einzelner Eintrag in der Maske korrigiert werden.

Die Maske selbst wird in den Textfeldern 156 und 157 gezeichnet. Das Wort MASK zeichnet die Maske auf den Bildschirm. .ADDR gibt den Inhalt eines Eintrags normal aus, .MAD zeichnet die Maske auf den Bildschirm und trägt den Inhalt eines Eintrags in diese ein. Textfeld 158 enthält das Menue des Adressenprogramms. Die Beschreibung des Programmablaufs ist in der Beschreibung des BUSIPACKS enthalten.

Im nächsten Textfeld ist das Wort zum Vergleichen von Zeichenketten programmiert. Dieses Wort wurde schon in Kapitel 3 gezeigt. Die nächsten Textfelder enthalten die Worte zur Eingabe und zur Ausgabe auf den Bildschirm oder Drucker. Wenn beim Suchen nach Einträgen ein zutreffender Eintrag gefunden wird, so wird der Begriff, nach dem gesucht wird, kurzzeitig verschoben und der gefundene Eintrag nach PAD geholt und von dort auf den Bildschirm ausgegeben. Danach wird der Suchbegriff wieder zurückgeholt.

Bei der Eingabe von mehreren Adressen wird durch das Wort MEHR die Eingabe der nächsten Adresse eingeleitet. Soll die Eingabe beendet werden, so ist das Wort ENDE einzugeben. Damit werden die Einträge

an das Ende der Liste gespeichert und FIRST# entsprechend erhöht. Wird der Eintrag mit N FINDE aus der Liste geholt, so muß er nach einer Änderung an die gleiche Stelle zurückgeschrieben werden.

Mit dem Wort LÖSCHEN wird ein Eintrag mit Leerzeichen aufgefüllt.

Die nächsten Textfelder enthalten die Worte für die Ausgabe auf einen Drucker. Abbildung 9.2 zeigt die Ausgabe auf Adressaufkleber.

```
                                     2                              1
   HANS PETER MUELLER                     VERLAG DER COMPUTERFREUNDE
   MEISTER                                ABT. RECHNUNGSWESEN
   LINDENWEG 2   8                        GRUENE BAUM STR. 27

 D-7500    KARLSRUHE                    D-8000   MUENCHEN
```

9.2 Adressaufkleber

Um zwei Etiketten gleichzeitig ausdrucken zu können, wird eine Zeile in PAD und PADD (PAD + RECLEN) zusammengebaut. Damit der Abstand stimmt, werden die auf den Eintrag folgenden Nullen, die durch EXPECT dort abgelegt wurden, durch SPAD entfernt.

Mit der Eingabe von

DRUCKER SUCHE ⟨BEZ⟩

werden Aufkleber gedruckt, welche die Suchbedingung erfüllen. Ist dabei schon eine Adresse für die Druckausgabe in PAD gespeichert, so wird diese kurzzeitig noch weiter nach oben geschoben. Die letzte Zeile der Adresse mit den beiden Codefeldern wird bei Aufklebern nicht, beim Drucken des Inhaltes jedoch mit ausgedruckt. Dies geschieht durch das Wort IDRUCK.

Die Worte SUCHE und INHALT kommen in unterschiedlichen Bedeutungen vor, je nachdem, ob sie auf den Drucker oder auf den Bildschirm ausgegeben werden. Deshalb sind zwei Wörterbücher TERMINAL und DRUCKER angelegt worden, um in diesen die Worte unterschiedlich definieren zu können. Das Wort RUN bringt nur das

Menue auf den Bildschirm. Das Wort NEU setzt FIRST# auf Eins und "löscht" somit die Liste. Dieses Wort muß am Anfang beim Erstellen einer Adressliste eingegeben werden.

Ab Textfeld 172 beginnt das Schreiben einer Rechnung. Das Produkt ist in Block 79 in Zeile 0 (Zeile 8 des Textfeldes 39) gespeichert. Zeile 1 enthält den Preis. Diese Angaben können mit dem Editor des FORTH Systems dort eingetragen werden. Außerdem enthält dieses Textfeld noch Angaben, die für das Schreiben einer Rechnung gebraucht werden. Dies zeigt Abbildung 9.3.

```
SCR # 39
  0 VERSANDKOSTEN
  1 RECHNUNGSBETRAG
  2 ZAHLBAR REIN NETTO 30 TAGE ZIEL
  3
  4
  5
  6
  7
  8 ELCOMP HEFT 4/5
  9 6.00
 10 RECHNUNG
 11 NUMMER
 12 % RABATT -
 13 MEHRWERTSTEUER
 14 5.00
 15
```

9.3 Textfeld 39 mit Angaben zum Schreiben einer Rechnung

Für den Preis und die Rechnungssumme werden doppelt lange Zahlen benutzt. Das Wort ABO leitet das Rechnungsschreiben ein. Es erscheint das Menue dieses Programmteils. Mit DATUM TTMMJJ wird das Datum eingegeben. DATUM 200384 wird auf der Rechnung als 20.03.84 ausgedruckt. RECHN# NNNN bestimmt die Nummer der nächsten Rechnung. Diese wird automatisch beim Schreiben von Rechnungen erhöht. Das Wort RES schreibt alle Rechnungen. Es beginnt bei der ersten Adresse und endet bei der letzten Adresse. Soll für nur einen Kunden eine Rechnung geschrieben werden, so geschieht

dies mit ⟨N⟩ REC. Dabei ist N die Nummer des Kunden, unter welcher er in der Adressliste zu finden ist.

In unserem Beispiel für den Versand einer Rechnung soll für jeden Kunden die Zahl der zu liefernden Hefte und der eingeräumte Rabatt vorgegeben werden. Für diese Einträge werden die Codefelder C1 und C2 in den Adressen benutzt. Im Codefeld C1 ist die Zahl der zu liefernden Hefte, im Codefeld C2 der Rabatt eingetragen. Das Wort P⟩Z, das die dort als Zeichenketten gespeicherten Zahlen in richtige Zahlen wandelt, benötigt als Begrenzer ein Leerzeichen nach der Zeichenkette. Für 100 Stück muß man 100 eingeben. Das gleiche gilt für den Rabatt. Für eine echte Applikation dieses Programmes wird man dieses Wort so ändern, daß das Leerzeichen automatisch eingefügt wird. Das Wort .VERLAG druckt die Firmenadresse aus. Diese ist als erste Adresse in der Liste gespeichert. Das Wort .KUNDE druckt die Anschrift des Kunden aus. Z1 schreibt das Wort Rechnung und Z2 die Rechnungsnummer und das Datum.

Das Wort MES benötigt drei Parameter auf dem Stapel. A ist die Blocknummer, N die Zeile, in welcher der auszugebende Text gespeichert ist. N' ist die Stelle, gerechnet vom linken Rand aus, ab welcher der Text ausgedruckt werden soll. Die Worte !ANZ und !RAB holen die Angaben aus den Codefeldern und speichern die Zahlen in den entsprechenden Variablen.

Für die Berechnung des Nettopreises und des Bruttopreises werden die Worte NET bzw. BRT benutzt. Beide Worte verwenden das Wort UDN*. Es multipliziert eine doppelt lange Zahl mit einer einfach langen Zahl. Als Ergebnis bleibt eine doppelt lange Zahl auf dem Stapel. Eine Implementation dieses Wortes in 6502 Maschinencode ist in Abbildung 9.4 angegeben.

Die Mehrwertsteuer ist als Konstante vereinbart. Die eingegebene Zahl bedeutet 7.0% Mehrwertsteuer. Für 14.0% ist die Konstante auf 1140 zu ändern. Das Wort Z3 druckt die Anzahl der Hefte, die Bezeichnung, den Bruttopreis und den Nettopreis aus. Danach berechnet es im SUM die Nettosumme aus. Falls ein Rabatt gewährt wurde, wird dieser berechnet und durch Z4 vom Nettobetrag abgezogen. Am Rechnungsende wird der Nettogesamtbetrag und die Mehrwertsteuer ausgewiesen.

```
SCR # 144
  0 ( FAW  UDN*                    EF)
  1 CODE UDN* ( DN-D')
  2 0 # LDY, BOT 4 + LDA, N STA,
  3 BOT 4 + STY, BOT 5 + LDA,
  4 N 1+ STA, BOT 5 + STY, SEC LDA,
  5 N 2+ STA, SEC STY, SEC 1+ LDA,
  6 N 3 + STA, SEC 1+ STY 16 # LDY,
  7 CS IF,
  8 N LDA, BOT 4 + ADC, BOT 4 + STA,
  9  N 1+ LDA, BOT 5 + ADC, BOT 5 +
 10 STA, N 2+ LDA, SEC ADC, SEC STA,
 11  N 3 + LDA, SEC 1+ ADC, SEC 1+
 12  STA,
 13 THEN, DEY, 0=
 14 UNTIL, INX, INX, NEXT JMP,
 15 END-CODE
 OK
```

9.4 Das Wort UDN*

Danach werden die Versandkosten berechnet. Diese sind ebenfalls in Block 79 eingetragen. Eine andere Möglichkeit, die Versandkosten zu berechnen ist folgende. In einem Block wird eine Tabelle der Versandkosten angelegt. Abhängig von der Stückzahl werden diese dann aus dieser Tabelle geholt und berechnet.

Das Wort 1RE schreibt eine Rechnung aus. Das Programm zeigt Abbildung 9.5, ein Beispiel ist in Abbildung 9.6 zu sehen.

```
SCR # 140
  0 ( START MAILING LIST VAR      EF)
  1 0 VARIABLE CNT 0 VARIABLE #NR
  2 : PRON 1 PR# ;
  3 : PROF 0 PR# ;
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13 -->
 14
 15
```

```
SCR # 141
  0 ( COMMON SCREENS BEGIN        EF)
  1  FORTH DEFINITIONS
  2  0 VARIABLE H 0 VARIABLE V
  3  : HO ( n) 0 DO SPACE 1 H +!
  4  LOOP ;
  5  : VE ( n) 0 DO CR   1 V +!
  6  0 H ! LOOP ;
  7 : ATYPE -DUP IF OVER + SWAP
  8   DO I C@ 127 AND DUP 0=
  9   IF DROP ELSE EMIT 1 H +!
 10  THEN  LOOP ELSE DROP ENDIF ;
 11 : PRINT ( NA) PAD + SWAP
 12   -TRAILING ATYPE ;
 13
 14      -->
 15
```

```
SCR # 142
  0  ( BUSI FORMAT                    ef)
  1  : ADJ ( n) H @ - -DUP IF HO
  2    THEN ;
  3  : PTEX> ( nan') ADJ PRINT ;
  4  : TEX>  ( ann') ADJ -TRAILING
  5    ATYPE ;
  6  : (RADJ) ( ann'n") SWAP ADJ
  7    SWAP DUP >R - HO R> ATYPE ;
  8  : #N ( d)    <# #S #> ;
  9  : #DM ( d) <# # # 46 HOLD #S
 10    #> ;
 11  : #% ( d) <# 37 HOLD # 46 HOLD
 12    #S #> ;
 13  : #DA ( d) <# # # 46 HOLD # #
 14    46 HOLD #S #> ;
 15   -->
```

```
SCR # 143
  0  (      VIRTUAL MEMORY         ef)
  1   80 CONSTANT START
  2   128 CONSTANT RECLEN
  3 : #INDEX ( n-a) RECLEN B/BUF
  4   */MOD START + DUP 356 = OVER
  5   356 > OR IF 2 +
  6   THEN BLOCK + ; ( ONLY C64)
  7  : FIRST# ( -a) 0 #INDEX ;
  8  : +NR 1 FIRST# +! UPDATE ;
  9  : !MEM PAD  FIRST# @ #INDEX
 10    RECLEN CMOVE UPDATE +NR ;
 11  : @MEM ( n) #INDEX PAD RECLEN
 12    CMOVE ;
 13 : !ENTRY PAD #NR @ #INDEX RECLEN
 14   CMOVE UPDATE FLUSH ;
 15   -->
```

```
SCR # 144
  0  ( BUSI DATUM INPUT          ef)
  1  0 VARIABLE DAT  6 ALLOT
  2  : DIN 13 WORD HERE COUNT ' DAT
  3  SWAP  CMOVE ;
  4  : D>S FIRST# 2 + ' DAT  SWAP 6
  5    CMOVE UPDATE ;
  6  : D<S FIRST# 2 + ' DAT  6
  7  CMOVE ;
  8
  9
 10
 11
 12
 13
 14 150 LOAD
 15
```

```
SCR # 150
  0 ( FORTH EXAMPLES               EF)
  1 : CU ( ZS) CH CV ;
  2 : DESCR 0 VARIABLE -2 ALLOT ;
  3 : NQ ' SPACE CFA ' QUIT 10 + ! ;
  4  : AQ ' CR CFA ' QUIT 10 + ! ;
  5
  6 : IP ( NA) PAD + SWAP EXPECT ;
  7 : (CL) ( N) 0 DO 32 EMIT LOOP ;
  8 : .- 45 EMIT ;
  9 : .— ( N) 0 DO .- LOOP ;
 10 : PAC ( NA) PAD + SWAP 32 FILL ;
 11
 12
 13         —>
 14
 15
```

```
SCR # 151
  0 ( FORTH EXAMPLES CNTD          EF)
  1 DESCR (NA)   0 , 28 ,
  2 DESCR (CO)  28 , 23 ,
  3 DESCR (ST)  51 , 23 ,
  4 DESCR (PLZ) 74 ,  7 ,
  5 DESCR (ORT) 81 , 21 ,
  6 DESCR (C1) 102 ,  5 ,
  7 DESCR (C2) 107 ,  7 ,
  8 DESCR (TEL) 114 , 11 ,
  9
 10          -->
 11
 12
 13
 14
 15
```

```
SCR # 152
  0 ( FORTH EXAMPLES CNTD          EF)
  1 : (NA (NA) 2@ ; : (CO (CO) 2@ ;
  2 : (ST (ST) 2@ ; : (PL (PLZ) 2@ ;
  3  : (OT (ORT) 2@ ;
  4 : (C1 (C1) 2@ ; : (C2 (C2) 2@ ;
  5 : (TEL (TEL) 2@ ;
  6
  7 : CNA 6 4 CU ; : CCO 8 4 CU ;
  8 : CST 10 4 CU ; : CPL 12 4 CU ;
  9 : COR 12 12 CU ; : CC1 14 4 CU ;
 10  : CC2 14 10 CU ;
 11 : CTE 14 18 CU ;
 12          -->
 13
 14
 15
```

```
SCR # 153
  0 ( FORTH EXAMPLES                    EF)
  1 : NAC CNA 28 (CL) (NA PAC ;
  2 : COC CCO 23 (CL) (CO PAC ;
  3 : STC CST 23 (CL) (ST PAC ;
  4 : CYC COR 21 (CL) (OT PAC ;
  5 : PLC CPL  7 (CL) (PL PAC ;
  6 : C1C CC1  5 (CL) (C1 PAC ;
  7 : C2C CC2  7 (CL) (C2 PAC ;
  8 : TEC CTE 11 (CL) (TEL PAC ;
  9
 10 : CL NAC COC STC CYC PLC C1C
 11   C2C TEC ;
 12         -->
 13
 14
 15
```

```
SCR # 154
  0 ( FORTH EXAMPLES CNTD               EF)
  1 : WAS 18 2 CU 30 (CL) 18 2 CU
  2   ." WAS" QUIT  ;
  3 : NA NAC CNA (NA IP WAS ;
  4 : CO COC CCO (CO IP WAS ;
  5 : ST STC CST (ST IP WAS ;
  6 : PLZ PLC CPL (PL IP WAS ;
  7 : ORT CYC COR (OT IP WAS ;
  8 : C1 C1C CC1 (C1 IP WAS ;
  9 : C2 C2C CC2 (C2 IP WAS ;
 10 : TEL TEC CTE (TEL IP WAS ;
 11 : EIN
 12   CNA (NA IP CCO (CO IP CST
 13   (ST IP CPL (PL IP COR (OT IP
 14   CC1 (C1 IP CC2 (C2 IP CTE (TEL
 15   IP WAS ;  -->
```

```
SCR # 155
  0 ( FORTH EXAMPLES CNTD        EF)
  1 : .CMD 20 0 CU 39 .— 21 1 CU
  2 ." EIN MEHR ENDE EXIT LOESCHEN M
  3 ENU"
  4   22 1 CU
  5 ." NA CO ST PLZ ORT C1 C2 TEL" ;
  6
  7 : 1S SPACE ; : 3S 3 SPACES ;
  8 : 5S 5 SPACES ;
  9
 10 —>
 11
 12
 13
 14
 15
```

```
SCR # 156
  0  ( BUSINESS MASK  cntd       ef)
  1  : .I 0 DO I 1+ 10 MOD   48 +
  2     EMIT   LOOP ;
  3 : 1R ( n) 3 CU 38 3 DO .- LOOP ;
  4  : 2R 6 3 CU .- 6 32 CU 6 .— ;
  5 : 3R ( n) DUP 3 CU .- 27 CU
  6   11 .— ;
  7  : 4R 12 3 CU .- 12 11 CU .-
  8   12 33 CU 5 .— ;
  9 : 5R 14 3 CU .- 14 9 CU .- 14 17
 10   CU .- 14 29 CU 6 .— 14 37 CU
 11    .- ;
 12  : 6R 16 4 CU 5 .I .- 7 .I
 13    .- 11 .I  .- ;
 14
 15                      —>
```

```
SCR # 157
  0  ( BUSINESS ADDR INPUT cntd  ef)
  1  : MASK CLR 5 1R 2R 7 1R 8 3R 9
  2    1R 10 3R 11 1R 4R 13 1R 5R
  3    15 1R 6R ;
  4  : .ADDR CR 3S (NA PRINT CR
  5 3S (CO PRINT CR 3S (ST PRINT CR
  6   3S (PL PRINT 1S (OT PRINT CR
  7 3S (C1 PRINT 1S (C2 PRINT CR
  8   3S (TEL PRINT ;
  9  : .MAD CLR MASK CNA (NA PRINT
 10  CCO (CO PRINT CST (ST PRINT
 11  CPL (PL PRINT COR (OT PRINT
 12  CC1 (C1 PRINT CC2 (C2 PRINT
 13  CTE (TEL PRINT ;
 14  : .MSG 2 10 CU   ." ADRESS-VERW
 15 ALTUNG" ;   -->
```

```
SCR # 158
  0  ( BUSINESS ADDR INPUT cntd  ef)
  1  : .MSG1 CLR .MSG 4 2 CU
  2  ." TERMINAL EINGABE"  5 11 CU
  3  ." SUCHE   <BZ> <NAME>" 6  5 CU
  4  ." <NR>  FINDE"  7 11 CU
  5  ." INHALT"
  6   10 2 CU
  7  ." DRUCKER  LABEL" 11 11 CU
  8  ." SUCHE   <BZ> <NAME>"
  9  12 11 CU ." INHALT "
 10  13 5 CU ." <NR>  DRUCKE"
 11
 12  15 2  CU ." RUN" 16 2 CU
 13  ." NEU " 17 2 CU ." ABO"
 14   18 2 CU ." WAS " QUIT ; -->
 15
```

```
SCR # 159
  0  ( BUSINESS SEARCH             ef)
  1  0 VARIABLE WO
  2 : ['] [COMPILE] ' ;
  3  : (VERGL) ( aa'c-f)
  4    BEGIN ROT DUP C@ >R OVER I =
  5          R> SWAP DUP IF 0
  6          ELSE DROP >R ROT DUP C@
  7          R> = DUP DUP THEN WHILE
  8          2DROP 1+ >R 1+ R> ROT
  9    REPEAT >R 2DROP 2DROP R> ;
 10  : WHAT ['] 2 - EXECUTE SWAP
 11   DROP DUP WO ! 13 WORD HERE
 12   COUNT ROT PAD + SWAP CMOVE ;
 13   -->
 14
 15
```

```
SCR # 160
  0  ( BUSINESS SEARCH  cntd     ef)
  1  : VERGL PAD WO @ + #NR @ #INDEX
  2          WO @ + 32 (VERGL) ;
  3  : NIL CR ." NICHT IN LISTE " ;
  4  : EOL CR ." ENDE DER LISTE " ;
  5  : .NAME #NR @ @MEM .ADDR ;
  6  : PADC PAD RECLEN 32 FILL ;
  7  : EINGABE MASK .CMD
  8    FIRST# @ DUP #NR ! 4 4 CU .
  9    WAS ;
 10
 11
 12
 13  -->
 14
 15
```

```
SCR # 161
  0  ( BUSINESS OUTPUT             ef)
  1  : 3? ( -f) CNT @ 3 = ;
  2  : (CONTENT) ( n)
  3    DUP . @MEM .ADDR
  4    1 CNT +! 3? IF KEY 0 CNT !
  5    CLR 32 = 1 XOR IF .MSG1
  6    THEN THEN ;
  7  : .CONTENT CLR    0 CNT !
  8     CR FIRST# @ DUP 1 = 1 XOR
  9    IF 1 DO I (CONTENT)
 10    CR LOOP THEN EOL KEY DROP
 11    .MSG1 ;
 12
 13
 14
 15  -->
```

```
SCR # 162
  0  ( BUSINESS SEARCHING cntd   ef)
  1  : MOVE> PAD PAD 128 + RECLEN
  2    CMOVE ;
  3  : <MOVE PAD 128 + PAD RECLEN
  4    CMOVE ;
  5  : FOUND MOVE> #NR @ (CONTENT)
  6     CR  <MOVE ;
  7  : (SEARCH) DO I #NR ! VERGL
  8   IF FOUND DROP 1 THEN LOOP ;
  9
 10  : SEARCH 0 CNT ! PADC CLR
 11    WHAT 0 FIRST# @ 1 (SEARCH)
 12    IF  EOL ELSE NIL THEN KEY
 13    DROP .MSG1 ;
 14  -->
 15
```

```
SCR # 164
  0  ( BUSINESS ENTRY                ef)
  1 : MEHR !MEM CL FIRST# @ 4 4 CU .
  2   WAS ;
  3 : ENDE !MEM FLUSH .MSG1 ;
  4 : EXIT !ENTRY .MSG1  ;
  5 : MENU .MSG1 ;
  6
  7
  8
  9  : (FINDE) ( n) DUP #NR ! DUP
 10    @MEM .MAD 4 4 CU .  ;
 11  : FINDE  (FINDE) .CMD WAS ;
 12
 13  : .* PAD RECLEN 32 FILL ;
 14 : LOESCHEN .* !ENTRY CL WAS ;
 15  -->
```

```
SCR # 166
  0  ( BUS. MAILING LABEL PRINT  ef)
  1  36 CONSTANT PH
  2   8 CONSTANT PV
  3 : PADD PAD     RECLEN + ;
  4  : TABS ( nn') SWAP - 0 DO SPACE
  5    LOOP ;
  6  : HTAB ( n)  PH TABS ;
  7  : (DR) ( naa'-n') + SWAP
  8  -TRAILING DUP ROT SWAP ATYPE ;
  9  : SPAD ( a)  DUP  RECLEN
 10    1 + + SWAP DO I C@ 0=
 11    IF 32 I  C! THEN LOOP ;
 12  : SS 2 SPACES ;
 13  : NCR ( n) 0 DO CR LOOP ;
 14  : 0Z 0 #N DUP >R ATYPE R> HTAB
 15    0 #N ATYPE ;    -->
```

```
SCR # 167
  0  ( BUSI. PRINTING cntd       ef)
  1  : 1Z PAD SPAD (NA PAD (DR)
  2    HTAB (NA PADD (DR) DROP ;
  3  : 2Z PAD SPAD (CO PAD (DR) HTAB
  4    (CO PADD (DR) DROP ;
  5  : 3Z PAD SPAD (ST PAD (DR) HTAB
  6    (ST PADD (DR) DROP ;
  7  : 4Z PAD SPAD (PL PAD (DR) 10
  8   TABS 10 (OT PAD (DR) + HTAB
  9   (PL PADD (DR) 10 TABS (OT
 10   PADD (DR) DROP ;
 11  : 5Z PAD SPAD (C1 PAD (DR) 1S
 12 1 + (C2 PAD (DR) + 1S 1 + (TEL
 13 PAD (DR) + HTAB (C1 PADD (DR) 1S
 14  (C2 PADD (DR) 1S (TEL PADD (DR)
 15  2DROP DROP ;  -->
```

```
SCR # 168
  0  ( BUSI. PRINTING cntd       ef)
  1  : DRUCKE SS 0Z CR SS 1Z CR  SS
  2     2Z  CR SS 3Z CR CR 4Z CR  ;
  3  : (MEM@) CNT @ DUP @MEM
  4     1 CNT +! ;
  5  : MEM@ ( a) BEGIN (MEM@) PAD C@
  6    42 = DUP IF SWAP  DROP THEN
  7    NOT UNTIL ;
  8  : FIN? ( -f) CNT @ FIRST# @ < ;
  9  : (LABEL)          BEGIN FIN?
 10  WHILE MEM@ MOVE> FIN? NOT IF 0
 11  PADC DRUCKE 3 NCR ELSE MEM@
 12  DRUCKE 3 NCR THEN REPEAT ;
 13  : .LABEL PRON 1 CNT !  (LABEL)
 14    PROF  .MSG1 ;  -->
 15
```

```
SCR # 169
  0  ( BUSI PRINT SEARCH            ef)
  1  : @MN #NR @ DUP @MEM 1 CNT +! ;
  2  : RC RECLEN ;
  3  : M> PAD PAD 256 + RC CMOVE ;
  4  : <M PAD 256 + PAD RC CMOVE ;
  5  : CNTO ( -f) CNT @ 2 MOD 0= ;
  6  : .SUCHE  M> @MN
  7     CNTO IF MOVE>  ELSE DRUCKE
  8     3 NCR THEN <M ;
  9  : ((SUCH)) DO I #NR ! VERGL
 10     IF .SUCHE THEN LOOP ;
 11
 12  : (SUCHE) 1 CNT ! PADC PRON
 13     WHAT 0 FIRST# @ 1 ((SUCH))
 14     CNTO IF 0 DRUCKE THEN
 15     PROF .MSG1 QUIT   ;  -->
```

```
SCR # 170
  0  ( BUSI .INHALT                 ef)
  1
  2  : LL 64 0 DO .- LOOP ;
  3
  4  : IDRUCK DRUCKE 5Z CR LL CR ;
  5  : (.INHALT) 1 CNT ! BEGIN FIN?
  6  WHILE MEM@ MOVE> FIN? NOT IF 0
  7  PADC IDRUCK ELSE MEM@ IDRUCK
  8  THEN REPEAT ;
  9  : .INHALT PRON  (.INHALT)
 10   PROF .MSG1 ;
 11
 12
 13
 14
 15  -->
```

```
SCR # 171
  0  ( BUSI VOCABULARY DRUCKER   ef)
  1  VOCABULARY DRUCKER IMMEDIATE
  2  DRUCKER DEFINITIONS
  3  : LABEL .LABEL ;
  4  : INHALT .INHALT ;
  5  : SUCHE (SUCHE) ;
  6  FORTH DEFINITIONS
  7  VOCABULARY TERMINAL IMMEDIATE
  8  TERMINAL DEFINITIONS
  9  : INHALT .CONTENT ;
 10  : SUCHE SEARCH ;
 11  FORTH DEFINITIONS
 12  : RUN CLR NQ    EMPTY-BUFFERS
 13  .MSG1 ;
 14  : NEU 1 FIRST# ! UPDATE FLUSH
 15  .MSG1 ;  -->
```

```
SCR # 172
  0 ( ABO                          EF)
  1 0 VARIABLE RE#
  2 0 VARIABLE SM 2 ALLOT
  3 0 VARIABLE ANZ 0 VARIABLE RAB
  4 0 VARIABLE VSK 2 ALLOT
  5 : ABO CLR 4 2 CU
  6    ." DATUM TTMMJJ" 5 2 CU
  7    ." RECHN# NNNN"  6 2 CU
  8    ." RES RECHNUNGEN SCHREIBEN"
  9    7 2 CU
 10   ." <N> REC RECHNUNG FUER KUNDE
 11 N SCHREIBEN"
 12    11 2 CU ." WAS" QUIT ;
 13 : #EIN ( -D) 13 WORD HERE DUP C@
 14  1+ + 32 SWAP C! HERE NUMBER ;
 15 -->
```

```
SCR # 173
  0 ( ABOVERWALTUNG START        EF)
  1 : .ADR
  2    (NA 10 PTEX> 1 VE
  3    (CO 10 PTEX> 1 VE
  4    (ST 10 PTEX> 1 VE
  5    (PL 10 PTEX> (OT 18 PTEX>
  6    1 VE ;
  7 : .VERLAG 1 @MEM 0 V ! 4 VE
  8   .ADR ;
  9 : .KUNDE ( N) @MEM 1 VE
 10   .ADR ;
 11 : DATUM DIN D>S FLUSH ABO ;
 12 : RECHN# #EIN DROP RE# ! ABO ;
 13 : (MES) ( AN-N') 32 * SWAP
 14   BLOCK + ;
 15 -->
```

```
SCR # 174
  0 ( ABOVERWALTUNG  CNTD        EF)
  1 : MES ( ANN') >R (MES) 32 R>
  2   TEX> ;
  3
  4 : .DAT ' DAT HERE 1+ 6 CMOVE
  5   HERE DUP 8 + 32 SWAP C!
  6   NUMBER #DA  67 TEX> ;
  7 : Z1 2 VE 79 2 27 MES ;
  8 : Z2 2 VE 79 3 10 MES RE# @ 0 #N
  9   18 TEX> .DAT ;
 10 : P>Z ( NA-D) PAD + SWAP HERE 1+
 11   SWAP CMOVE HERE NUMBER ;
 12 : ANZ! (C1 P>Z DROP ANZ ! ;
 13 : RAB! (C2 P>Z DROP RAB ! ; -->
 14
 15
```

```
SCR # 175
  0 ( ABOVERWALTUNG CNTD          EF)
  1 1070 CONSTANT MWST
  2 : NET ( DN-D') >R 1000 UDN*
  3   R M/MOD ROT R> 2 / 1 - > IF
  4   1. D+ THEN ;
  5
  6 : BRT ( DN-N') UDN* 1000 M/MOD
  7   ROT 499 > IF 1. D+ THEN ;
  8
  9 : .ANZ ANZ @ 0 #N 15 6 (RADJ) ;
 10 : .BZ 79 0 32 MES ;
 11 : .PR 79 1 (MES) 5 52 8 (RADJ) ;
 12  : .NPR 79 BLOCK 31 + NUMBER
 13   MWST NET 2DUP SM 2! #DM 61 8
 14   (RADJ) ;
 15 -->
```

```
SCR # 176
  0 ( RECHNUNGSCHREIBEN CNTD      EF)
  1 : SUM SM 2@ ANZ @ UDN* SM 2! ;
  2 : .SU SM 2@ #DM 69 9 (RADJ) ;
  3
  4 : Z3 5 VE ANZ! .ANZ .BZ .PR .NPR
  5   SUM .SU ;
  6
  7 0 VARIABLE SME 2 ALLOT
  8 : RABATT RAB! SM 2@ RAB @ UDN*
  9   100 M/MOD ROT 49 > IF 1. D+
 10   THEN SME 2! SM 2@ SME 2@
 11   DMINUS D+ SM 2! ;
 12 : .RA 53 ADJ RAB @ 0 #N ATYPE
 13   79 4 55 MES SME 2@ #DM 69 9
 14   (RADJ) ;
 15 -->
```

```
SCR # 177
  0 ( RECHNUNGSCHREIBEN CNTD     EF)
  1 : Z4 RABATT 2 VE .RA ;
  2
  3 : .EPR SM 2@ #DM 69 9 (RADJ) ;
  4 : .BT ( DN) >R #DM R> 9 (RADJ) ;
  5
  6 : .MWST 79 5 32 MES 1000 - 0 #%
  7   60 6 (RADJ) ;
  8 : Z5 5 VE .EPR ;
  9
 10 : MW1 SM 2@ MWST BRT 2DUP SME 2!
 11   SM 2@ DMINUS D+ 69 .BT ;
 12 : Z6 1 VE MWST .MWST MW1 ;
 13
 14 -->
 15
```

```
SCR # 178
  0 ( RECHNUNGSSCHREIBEN CNTD    EF)
  1 : .VSK 79 BLOCK 191 + NUMBER
  2   2DUP VSK 2! MWST NET 69 .BT ;
  3
  4 : Z7 1 VE 78 0 32 MES .VSK ;
  5 : Z9 1 VE 78 1 32 MES
  6  VSK 2@ SME 2@ D+ 69 .BT ;
  7
  8 : MW2 VSK 2@ 2DUP MWST NET
  9   DMINUS D+ 69  .BT ;
 10 : Z8 1 VE MWST .MWST MW2 ;
 11 : Z10 1 VE 78 2 20 MES ;
 12 : 1RE ( N) .VERLAG .KUNDE Z1 Z2
 13   Z3 2 VE Z4 Z5 Z6 Z7 Z8 Z9 Z10
 14   75 V @ - VE 1 RE# +! ;
 15
```

```
SCR # 179
  0 ( RECHNUNGSSCHREIBEN ENDE    EF)
  1
  2 : REC ( N) PRON 1RE PROF ;
  3
  4 : RES PRON FIRST# @ 0 DO I 1RE
  5   LOOP PROF MENU ;
  6
```

9.5 Programm Adressverwaltung mit Rechnungsschreiben

```
VERLAG DER COMPUTERFREUNDE
ABT. RECHNUNGSWESEN
GRUENE BAUM STR. 27
D-8000  MUENCHEN

HANS PETER MUELLER
MEISTER
LINDENWEG 28
D-7500  KARLSRUHE  ;


                 RECHNUNG

NUMMER   1002                                        3.04.84



         100          ELCOMP HEFT 4/5       6.00       5.61    561.00


                                       25% RABATT -            140.25



                                                               420.75
                      MEHRWERTSTEUER                 7.0%       29.45
                      VERSANDKOSTEN                              4.67
                      MEHRWERTSTEUER                 7.0%        0.33
                      RECHNUNGSBETRAG                          455.20
           ZAHLBAR REIN NETTO 30 TAGE ZIEL
```

9.6 Beispiel

Das Programm enthält folgende systemabhängige Worte:

PRON Schaltet Drucker ein.
PROF Schaltet Drucker aus.
CU (ZS) Setzt Cursor in die Zeile Z und Spalte S.
CLR Löscht Bildschirm.

Das Wort .MSG1 wird mit QUIIT verlassen. Dadurch wird kein OK ausgegeben, aber ein CR. Das Wort NQ ersetzt in QUIT die Codefeldadresse von CR durch die Codefeldadresse von SPACE. Dadurch bleibt der Cursor hinter WAS stehen. An der verwendeten FORTH-Version ist zu prüfen, ob der Abstand zwischen der Parameterfeldadresse von QUIT und dem Eintrag der Codefeldadresse genau 10 ist. Wenn nicht, so ist diese Zahl in NQ und AQ zu ändern. AQ stellt den alten Zustand von QUIT wieder her.

Notizen

10 Ein kleines Spielchen

Das folgende kleine Spiel "BRAIN TEASER" sieht auf den ersten Blick harmlos aus. Es bedarf jedoch einiger Überlegung, um mit sechs Zügen die Lösung zu finden. Auf dem Bildschirm wird eine 3 x 3 Matrix, die zufällig mit Nullen und Einsen gefüllt ist, ausgegeben.

Beispiel:

```
1 0 1        1 2 3
0 0 0        4 5 6
0 1 0        7 8 9
```

Die nebenstehende Matrix zeigt die Bezeichnung der Elemente. Aus dieser Anfangssituation soll durch Ändern der Nullen und Einsen die Lösung

```
1 1 1
1 0 1
1 1 1
```

erreicht werden.

Dazu stehen folgende Befehle zur Verfügung:

Wird das mittlere Element 5 vertauscht, so ändern sich auch die Elemente 2, 4, 6 und 8. Aus

```
1 0 1          1 1 1
0 0 0   wird   1 1 1
0 1 0          0 0 0
```

Wird ein Elemment in einer Ecke (1 3 7 9) vertauscht, so ändern sich auch die drei benachbarten Elemente. Wird das Element 1 geändert, so erhält man aus

```
1 0 1        0 1 1
0 0 0   →    1 1 0
0 1 0        0 1 0
```

Wird ein Element in der Mitte einer Zeile oder Spalte (2 4 6 8) geändert, so ändern sich auch die beiden anderen Elemente in dieser Zeile oder Spalte. Wird Element 8 geändert, so erhält man aus

```
1 0 1        1 0 1
0 0 0   →    0 0 0
0 1 0        1 0 1
```

Werden beim Vertauschen alle Elemente Null, so hat man verloren.

Im Programm in Abbildung 10.1 ist jedem Element der Matrix ein Bit der obersten Zahl im Stapel zugeordnet. Bit 1 entspricht Element 1, Bit 2 entspricht Element 2 usw.

Das Wort DR zeichnet eine Zeile der Matrix. Durch 2 / MOD SWAP . wird der Rest der Division mit 2 auf den Bildschirm ausgegeben. Dies wird in einer Schleife dreimal ausgeführt.

Das Wort DRAW zeichnet die Matrix auf den Bildschirm. Dieses Wort verwendet das Wort CU. CU ist rechnerspezifisch und setzt den Cursor auf dem Bildschirm. Die oberste Zahl auf dem Stapel ist die horizontale, die zweite Zahl ist die vertikale Position auf dem Bildschirm.

Neben der Matrix wird noch die Bezeichnung der Elemente ausgegeben. Enthält die neue Matrix nur Nullen, so hat man verloren.

Für das Vertauschen der Elemente wird die EXCLUSIV-ODER Funktion verwendet.

A B XOR ergibt

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Ist A gleich 0, so bleibt in B das Bit erhalten, ist A gleich 1, so wird der Inhalt von B vertauscht.

Wird also Element 5 geändert, so ändern sich auch die Elemente 2 4 6 und 8. Das ergibt folgendes Bitmuster:

```
0 1 0
1 1 1
0 1 0
```

Bit 1 ist das am weitesten rechts stehende Bit.
Die Zahlen 1 bis 9 bilden die EXCLUSIV-ODER Funktion mit dem obersten Wort aus dem Stapel und zeichnen das neue Bitmuster auf dem Bildschirm.

Vorsicht, wenn man später mit anderen Worten arbeitet, denn die Zahlen 1 bis 3 sind Konstante, die im Kernel definiert sind.

Die Zeilen 7 bis 10 sind ein Zufallsgenerator. In GAME wird eine Zufallszahl kleiner 512 erzeugt und diese als Bitmuster ausgegeben.

Durch Eingabe der Nummer eines Elementes werden die entsprechenden Elemente vertauscht. Werden dabei alle Elemente Null, so hat man verloren. Ein Beispiel zeigt Abbildung 10.2.

```
SCR # 117
  0 ( FAW BRAIN TEASER              EF)
  1 : DR ( N-N') 3 0 DO 2 /MOD
  2   SWAP . LOOP ;
  3 : DRAW ( N-N) DUP
  4 10 10 CU DR 10 20 CU ." 1 2 3"
  5 11 10 CU DR 11 20 CU ." 4 5 6"
  6 12 10 CU DR 12 20 CU ." 7 8 9"
  7 DROP DUP 511 AND 0= IF
  8 CR ." VERLOREN" DROP THEN ;
  9 2 BASE !
 10 : 5 010111010 XOR DRAW ;
 11 : 1 000011011 XOR DRAW ;
 12 : 3 000110110 XOR DRAW ;
 13 : 7 011011000 XOR DRAW ;
 14 : 9 110110000 XOR DRAW ;
 15 -->
```

```
SCR # 118
  0 ( FAW BRAIN TEASER CNTD         EF)
  1 : 2 000000111 XOR DRAW ;
  2 : 4 001001001 XOR DRAW ;
  3 : 6 100100100 XOR DRAW ;
  4 : 8 111000000 XOR DRAW ;
  5
  6 DECIMAL
  7 0 VARIABLE RND HERE RND !
  8 : RANDOM RND @ 31421 * 6972 +
  9    DUP RND ! ;
 10 : RNDNR RANDOM U* SWAP DROP ;
 11 : GAME CLR 512 RNDNR DRAW ;
 12
 13
 14
 15 ;S
```

10.1 BRAIN TEASER

```
GAME
1 1 0     1 2 3
0 0 1     4 5 6
1 1 1     7 8 9

 OK
1
0 0 0     1 2 3
1 1 1     4 5 6
1 1 1     7 8 9

 OK
5
0 1 0     1 2 3
0 0 0     4 5 6
1 0 1     7 8 9

 OK
2
1 0 1     1 2 3
0 0 0     4 5 6
1 0 1     7 8 9

 OK
4
0 0 1     1 2 3
1 0 0     4 5 6
0 0 1     7 8 9

 OK
6
0 0 0     1 2 3
1 0 1     4 5 6
0 0 0     7 8 9

 OK
5
0 1 0     1 2 3
0 1 0     4 5 6
0 1 0     7 8 9

 OK
8
0 1 0     1 2 3
0 1 0     4 5 6
1 0 1     7 8 9

 OK
2
1 0 1     1 2 3
0 1 0     4 5 6
1 0 1     7 8 9

 OK
5
1 1 1     1 2 3
1 0 1     4 5 6
1 1 1     7 8 9

 OK
```

10.2 Ein Beispiel

Notizen

11 Der 6502 Assembler von W.F. Ragsdale

Der im folgenden beschriebene Assembler für 6502 Systeme ist in FORTH DIMENSIONS VIII, NR5, PP 143–150 veröffentlicht. Er wurde von W. F. Ragsdale der Öffentlichkeit frei zur Verfügung gestellt. Diese Beschreibung ist eine gekürzte, freie Übersetzung der obigen Veröffentlichung.
Einige Beispiele wurden zusätzlich mit aufgenommen.

Dieser Assembler besitzt folgende Eigenschaften:

1. Vom Benutzer können jederzeit MACROS definiert werden.
2. Zahlenangaben können in einer beliebigen Zahlenbasis gemacht werden.
3. In den Ausdrücken können alle Rechnungsarten verwendet werden.
4. Zur Programmierung werden strukturierte Schleifen mit bedingten Anweisungen verwendet. Es sind keine Marken zugelassen.
5. Der Assembler selbst ist in FORTH geschrieben und belegt 1300 Byte Object Code.

Bei der Assemblierung enthält die Variable CONTEXT die Wörterbuchadresse des Assemblers. Worte aus dem Eingabespeicher werden zuerst in Assembler, dann in FORTH, in den vom Benutzer aufgestellten Wörterbüchern und zuletzt im FORTH Kernel gesucht. Wird das Wort nicht gefunden, wird untersucht, ob es sich um eine Zahl handelt. Ist dies nicht der Fall, so wird mit einer Fehlermeldung abgebrochen. Dies ist die übliche Reihenfolge, mit welcher der FORTH-Interpreter Worte in das Wörterbuch aufnimmt.

Während der Assemblierung einer CODE-Definition werden die Worte in ausführbaren Maschinencode übersetzt. Dabei werden für die Ver zweigungen die nötigen Adressrechnungen durchgeführt. Die bedingten Anweisungen wie IF, BEGIN, UNTIL werden ebenfalls in Adressen oder Maschinencode übersetzt.

Während der Laufzeit des Programms, wenn also die CODE Worte auf gerufen werden, wird dieser erzeugte Maschinencode durchlaufen. Dies geschieht unter der Kontrolle des Interpreters. Das Ende eines CODE Wortes ist also kein RTS, wie in Maschinensprache üblich, sondern ein absoluter Sprung nach NEXT. Dies ist für den inneren Interpreter die Aufforderung, das nächste FORTH Wort zu verarbeiten. Da in Ma schinencode die Register der CPU benötigt werden. Dies gilt besonders für das X-Register, das in 6502 FORTH als Stapelzeiger verwendet wird. Dafür ist im Kernel eine feste Adresse XSAVE reserviert.

Die meisten Assemblerworte sind mit einem "," abgeschlossen. Diese Schreibweise hat drei Gründe:

1. Das Komma zeigt den logischen Abschluß einer Anweisung an und entspricht somit einer Zeile in Assemblerschreibweise.
2. In FORTH speichert das Komma eine Zahl im Wörterbuch. Somit wird durch das Komma angezeigt, das diese Anweisung im Wörter buch gespeichert wird.
3. Das Komma unterscheidet bestimmte Befehlscode von möglichen Hexzahlen, wie ADC oder ADD.

Der Assembler führt mehrere Tests auf Fehler in der Eingabe aus.

1. Alle bedingten Anweisungen müssen richtig geschachtelt und ge paart sein.
2. Die Adressierungsarten und Operanden müssen bei den Befehls codes erlaubt sein.
3. Alle Parameter, die in CODE benutzt werden, müssen entfernt werden

Diese Tests werden durch Überwachung des Stapels und durch Bitmasken für die Adressierungsarten durchgeführt.

Wird ein Fehler gefunden, so wird das Wort END-CODE nicht durchlaufen und die Definition bleibt unsichtbar im Wörterbuch.

Die Fehlermeldung DEFINITION NOT FINISHED erscheint, wenn der Wert der USER Variablen beim Ende der Definition vom Wert des Stapelzeigers abweicht. Die Fehlermeldung CONDITIONALS NOT PAIRED wird ausgegeben, wenn Fehler bei der Verschachtelung auftreten. Ist bei einem Befehlscode die angegebene Adressierung nicht möglich, so erscheint die Meldung HAS INCORRECT ADRESS MODE.

Schreibweise der Adressierungsarten

Für die einzelnen Adressierungsarten sind folgende Schreibweisen festgelegt:

#	unmittelbar	nur 8 Bit
,X	mit X indiziert	Zero-Page absolut
,Y	mit Y indiziert	Zero-Page absolut
X)	indiziert indirekt	nur Zero-Page
)Y	indirekt indiziert	nur Zero-Page
	Speicher	Zero-Page absolut

Es folgen einige Beispiele in FORTH-Assembler und normaler Assembler Schreibweise:

```
        .A  ROL,   ROL A oder ROL
       1 #  LDY,   LDY #1
   DATA ,X  STA,   STA DATA,X
   DATA ,Y  CMP,   CMP DATA,Y
      6 X)  ADC,   ADC (6,X)
  POINT )Y  STA,   STA (POINT),Y
VECTOR  )   JMP,   JMP (VECTOR)
```

Einige Besonderheiten bei der Implementierung von FORTH auf 6502 Systemen.

Der Datenstapel wird in der Zero-Page angelegt und wächst zu niederen

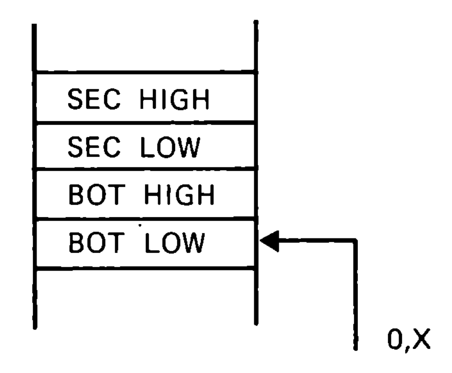
Adressen. Die Anfangsadresse ist rechnerabhängig. Das X-Register wird als Stapelzeiger verwendet. Ein Element des Stapels ist ein 16-Bit Wort und belegt somit 2 Byte. Die Befehlsfolge INX, INX, entspricht dem Wort DROP. Um auf das unterste und das zweite Wort auf dem Stapel zugreifen zu können, sind im Assembler zwei Worte BOT und SEC vereinbart.

```
BOT LDA, entspricht LDA (0,X)
SEC LDY, entspricht LDY (2,X)
```

Mit

```
BOT LDA
BOT 1+ LDA,
```

werden die niederwertigen 8 Bit des obersten Elementes auf dem Stapel im Akkumulator und die höherwertigen 8 Bit im Y-Register gespeichert. Die folgende Zeichnung zeigt die Bezeichnung für die obersten zwei Elemente des Stapels.



Der Prozessor Stapel der 6502 CPU wird in der üblichen Weise verwendet. Das Wort RP) bringt die Adresse des tiefsten Elementes auf den Datenstapel. Dort ist das zuletzt auf dem Return-Stapel abgelegte Byte gespeichert. Denn auch dieser Stapel wächst zu niedrigeren Adressen.

Bedingte Verzweigungen

Der Assembler benötigt keine Marken. Diese werden durch zwei Besonderheiten ersetzt. Jedes in FORTH definierte Wort kann jederzeit

in einer CODE Definition verwendet werden.

Zweitens wird die Assemblierung durch bedingte Verzweigungen

BEGIN, ... cc UNTIL,

und

cc IF, . . . ELSE, . . . THEN,

gesteuert. Die Abkürzung cc bedeutet Condition Code. Dieser Code steuert die Verzweigung.

Das folgende Beispiel zeigt die Programmierung einer Warteschleife.

```
CODE DEMO1
   128 # LDA,
     N STA,
BEGIN,
    N DEC,
    0=
UNTIL,
NEXT JMP, END-CODE
```

Die Hilfszelle N wird in einer BEGIN, . . . UNTIL, Schleife dekrementiert. Als Bedingung für den Schleifenabbruch wird die Abfrage 0= verwendet. Wenn diese Bedingung erfüllt ist, wird das nächste Wort interpretiert. Das Programm erzeugt folgenden Code.

```
35C8 52  8 85 44 45 4D 4F B1   R..DEMO1
35D0 B5 35 D4 35 A9 80 85 46   55T5)..F
35D8 C6 46 D0 FC 4C 52  8  4   FFP LR..
```

Das Wort BEGIN, legt bei der Assemblierung die augenblickliche Adresse und eine Eins auf dem Stapel ab. Das Wort UNTIL benutzt nun bei der Assemblierung diese Adresse für einen bedingten, relativen

Sprung zurück ins Programm. Die Art der Verzweigung wird durch c
bestimmt. Die von BEGIN ebenfalls abgelegte Eins wird zur Fehler
prüfung verwendet.

Als Bedingungen für die Verzweigung können folgende Abfragen ein
gesetzt werden:

- 0= Verzweigung, wenn Inhalt von Register oder Zelle Null ist.
- 0< Verzweigung, wenn Inhalt von Register oder Zelle kleiner Null ist.
- CS Verzweigung, wenn das CARRY-Bit gesetzt ist.

Das Wort DEMO ist ein Beispiel für eine Verzweigung mit IF, ELSE, THEN, .

```
HEX
CC00 CONSTANT PORTA
CODE DEMO
  PORTA LDA,
     0< IF,
     DEY,
     ELSE,
     INY,
     THEN,
NEXT JMP, END-CODE
DECIMAL
```

Je nachdem, ob der eingelesene Wert kleiner oder größer Null ist, wird das Y-Register erniedrigt oder erhöht.
Das nächste Beispiel zeigt die Verschachtelung von bedingten Verzweigungen.

```
CODE DEMO2
BEGIN
 PORTA LDA,
 0= IF, N INC,
    ELSE, N DEC,
    THEN,
 N LDA,
 0< UNTIL,
NEXT JMP, END-CODE
```

Wenn sich die Verzweigungen überlappen, wird eine Fehlermeldung ausgegeben.

Nicht immer kann die Strukturierung durchgehalten werden. Dann muß die Fehlerprüfung des Assemblers überlistet werden. Dies wird im Anschluß an die Beschreibung des Assemblers im Beispiel Analog-Digital Wandlung beschrieben.

Über den Assembler kann auf einige Einsprünge im Kernel zugegriffen werden, um Daten vom Maschinenprogramm auf den Stapel zu legen oder von dort zu holen.

Mit PUSH wird ein 16-Bit Wert auf dem Datenstapel abgelegt.

```
CODE DEMO3
 PORTA LDA,
       PHA,
   0 # LDA,
  PUSH JMP, END-CODE
```

Das niederwertige Byte wird über den RETURN-Stapel, das höherwertige Byte wird über den Akkumulator, an PUSH übergeben.

Mit den gleichen Bedingungen kann eine 16-Bit Zahl auch an PUT übergeben werden. PUT überschreibt mit diesen Werten das oberste Element des Datenstapels.

Anstatt mit PUSH, kann man auch mit folgendem Programm Daten auf dem Stapel ablegen.

```
CODE DEMO4
 PORTA LDA,
  DEX, DEX,
   BOT STA,
   0 # LDA,
BOT 1+ STA,
  NEXT JMP, END-CODE
```

Werden in einem Maschinenprogramm Daten vom Stapel entnommen, so kann durch JMP POP der Stapelzeiger um ein, durch JMP POPTWO um zwei 16-Bit Worte erniedrigt werden. Beide Einsprünge führen auf das Wort NEXT.

```
SCR # 200
  0 ( 6502 ASSEMBLER BY W.F.RAGSDALE
  1 FORTH DIMENSIONS VOL 3 P 143
  2 PUBLIC DOMAIN )
  3 HEX
  4 VOCABULARY ASSEMBLER IMMEDIATE
  5 ASSEMBLER DEFINITIONS
  6 ( SYSTEM SPECIFIC CONSTANTS )
  7 55 CONSTANT XSAVE 51 CONSTANT W
  8 53 CONSTANT UP    4E CONSTANT IP
  9 46 CONSTANT N
 10 ' (DO) 0E + CONSTANT POP
 11 ' (DO) 0C + CONSTANT POPTWO
 12 ' LIT 13 + CONSTANT PUT
 13 ' LIT 11 + CONSTANT PUSH
 14 ' LIT 18 + CONSTANT NEXT
 15  -->
```

```
SCR # 201
  0 ( ASSEMBLER CNTD                      )
  1 ' EXECUTE NFA 11 - CONSTANT
  2   SETUP
  3 0 VARIABLE INDEX -2 ALLOT
  4 0909 , 1505 , 0115 , 0811 ,
  5 8009 , 1D0D , 8019 , 8080 ,
  6 0080 , 1404 , 8014 , 8080 ,
  7 8080 , 1C0C , 801C , 2C80 ,
  8 2 VARIABLE MODE
  9 : .A 0 MODE ! ; : # 1 MODE ! ;
 10 : ,X 3 MODE ! ; : ,Y 4 MODE ! ;
 11 : )Y 6 MODE ! ; : ) F MODE ! ;
 12 : MEM 2 MODE ! ; : X) 5 MODE ! ;
 13
 14 : BOT ,X 0 ; : SEC ,X 2 ;
 15 : RP) ,X 101 ; -->
```

```
SCR # 202
  0 ( ASSEMBLER CNTD                          )
  1 : UPMODE IF MODE @ 8 AND 0= IF
  2   8 MODE +! THEN THEN 1 MODE @
  3   0F AND -DUP IF 0 DO DUP + LOOP
  4   THEN OVER 1+ @ AND 0= ;
  5
  6 : CPU <BUILDS C, DOES> C@ C,
  7   MEM ;
  8
  9 00 CPU BRK,    18 CPU CLC,
 10 D8 CPU CLD,    58 CPU CLI,
 11 B8 CPU CLV,    CA CPU DEX,
 12 88 CPU DEY,    E8 CPU INX,
 13 C8 CPU INY,    EA CPU NOP,
 14 48 CPU PHA,    08 CPU PHP,
 15 68 CPU PLA,    28 CPU PLP, -->
```

```
SCR # 203
  0 ( ASSEMBLER CNTD                          )
  1 40 CPU RTI,    60 CPU RTS,
  2 38 CPU SEC,    F8 CPU SED,
  3 78 CPU SEI,    AA CPU TAX,
  4 A8 CPU TAY,    BA CPU TSX,
  5 8A CPU TXA,    9A CPU TXS,
  6 98 CPU TYA,
  7
  8 : MU <BUILDS C, , DOES>
  9   DUP 1+ @ 80 AND IF 10 MODE +!
 10   THEN OVER FF00 AND UPMODE
 11   UPMODE IF MEM CR LATEST ID.
 12   3 ERROR THEN C@ MODE C@ INDEX
 13   + C@ + C, MODE C@ 7 AND IF
 14   MODE C@ 0F AND 7 < IF C, ELSE
 15   , THEN THEN MEM ; -->
```

```
SCR # 204
  0 ( ASSEMBLER CNTD                          )
  1 1C6E 60 MU ADC, 1C6E 20 MU AND,
  2 1C6E C0 MU CMP, 1C6E 40 MU EOR,
  3 1C6E A0 MU LDA, 1C6E 00 MU ORA,
  4 1C6E E0 MU SBC, 1C6E 80 MU STA,
  5 0D0D 01 MU ASL, 0C0C C1 MU DEC,
  6 0C0C E1 MU INC, 0D0D 41 MU LSR,
  7 0D0D 21 MU ROL, 0D0D 61 MU ROR,
  8 0414 81 MU STX, 0486 E0 MU CPX,
  9 0486 C0 MU CPY, 1496 A2 MU LDX,
 10 0C8E A0 MU LDY, 048C 80 MU STY,
 11 0480 14 MU JSR, 8480 40 MU JMP,
 12 0480 20 MU BIT,
 13
 14    -->
 15
```

```
SCR # 205
  0 ( ASSEMBLER CNTD                          )
  1 : BEGIN, HERE 1 ; IMMEDIATE
  2 : UNTIL, ?EXEC >R 1 ?PAIRS R>
  3   C, HERE 1+ - C, ; IMMEDIATE
  4 : IF, C, HERE 0 C, 2 ; IMMEDIATE
  5  : THEN, ?EXEC 2 ?PAIRS HERE
  6    OVER C@ IF SWAP ! ELSE OVER
  7    1+ - SWAP C! THEN ; IMMEDIATE
  8  : ELSE, 2 ?PAIRS HERE 1+ 1 JMP,
  9    SWAP HERE OVER 1+ - SWAP C!
 10    2 ; IMMEDIATE
 11 : NOT 20 + ;
 12 90 CONSTANT CS D0 CONSTANT 0=
 13 10 CONSTANT 0< 90 CONSTANT >=
 14    -->
 15
```

```
SCR # 206
  0 ( ASSEMBLER END                        )
  1 : END-CODE CURRENT @ CONTEXT !
  2   ?EXEC ?CSP SMUDGE ; IMMEDIATE
  3
  4 FORTH DEFINITIONS DECIMAL
  5
  6 : CODE ?EXEC CREATE [COMPILE]
  7   ASSEMBLER ASSEMBLER MEM !CSP ;
  8 -->
  9
 10
 11
 12
 13
 14
 15
```

Wörterbuch des Assemblers:

.X indizierte Adressierung mit X

.Y indizierte Adressierung mit Y

0< (– cc) (assemblieren)

Verzweigung, wenn das Negativ-Bit im Statusregister Eins ist. Während des Assemblierens wird cc auf dem Stapel abgelegt.

0 (–cc) (assemblieren)

Verzweigung, wenn das Zero Bit im Status Register Eins ist. Während des Assemblierens wird cc auf dem Stapel abgelegt.

;CODE wird in einer Doppelpunkt-Definition verwendet, um von der Programmierung in high level in Assembler umzuschalten.

Beispiel:

: <NAME> <WORT1> . . . . ; CODE
<ASSEMBLER> END-CODE

ASSEMBLER (in FORTH)

setzt CONTEXT auf Assembler.

BEGIN, (–A 1) (assemblieren)

Während des Assemblierens ist A eine Adresse, bei der eine Schleife beginnt. Die Eins wird zur Prüfung der bedingten Verzweigung benötigt.

BOT (–N) Relative Adresse des niederwertigen Bytes des obersten Elementes auf dem Stapel. Wird beim Assemblieren in die indiziert indirekte Adressierung (0,X) gewandelt.

CODE Ein Definitionswort, das den Beginn eines Wortes, geschrieben in Assembler anzeigt.

CPU (N) (compilieren)

Ein Definitionswort des Assemblers, das Befehle ohne Adressierung (0,X) erzeugt.

Beispiel EA CPU NOP,

CS (–cc) (assemblieren)

Verzweigung, wenn das CARRY Bit gesetzt ist.

ELSE, (A1 2 –A2 2) (assemblieren)

Der Programmteil hinter ELSE wird während der Laufzeit ausgeführt, wenn cc vor IF, FALSE ist. Beim Assemblieren wird ein Vorwärtssprung nach THEN, ausgeführt. Die Zwei auf dem Stapel wird für die Fehlererkennung benötigt.

END-CODE Beendet eine CODE-Definition. Dabei findet eine Fehlerprüfung statt. Ist kein Fehler aufgetreten wird das Wort in das CURRENT Wörterbuch eingetragen. Mit SMUDGE wird es zur Ausführung bereitgestellt.

IF, (cc– A 2)

Während des Assemblierens wird ein bedingter Vorwärtssprung, der von cc abhängt, erzeugt. Die Adresse A bleibt für das zugehörige ELSE, oder THEN, auf dem Stapel. Die Zwei wird für die Fehlerprüfung benötigt.

INDEX (–A)
Eine Matrix aus Bitmustern, welche die für einen Befehlscode möglichen Adressierungsarten prüft.

IP (–A) (assemblieren)
Während des Assemblierens ist IP eine Konstante, mit der Adresse des Instruction Pointers.

Während der Laufzeit enthält IP die Adresse des nächsten Wortes, das von NEXT aufgerufen wird.

M/CPU (N1N2) (kompilieren)
Ein Definitionswort des Assemblers. Es erzeugt Operationscodes mit mehrfachen Möglichkeiten der Adressierung. Die beiden Parameter sind der Op-Code und das Bitmuster für die Prüfung der zulässigen Adressierung.

MEM setzt MODE auf absolute Adressierung, Zero-Page.

MODE (–a) Variable, welche die augenblickliche Adressierungsart enthält.

N (–a) (assemblieren)
Adresse eines frei verfügbaren kleinen Speicherbereichs in der Zero-Page.

NEXT (–a) (assemblieren)
Einsprungsadresse für die Ausführung von NEXT.

NOT (F–F') Invertieren des cc.

POP (–A) (assemblieren)
(N) (Laufzeit)
Während des Assemblierens eine Konstante, welche die Einsprungsadresse für POP enthält.
Während der Laufzeit wird die oberste Zahl vom Stapel entfernt.

POPTWO (—a) (assemblieren)
(NN') (Laufzeit)

Während des Assemblierens eine Konstante der Einsprungadresse POPTWO.
Während der Laufzeit werden die beiden obersten Elemente des Stapels entfernt.

PUSH (—a) (assemblieren)
(—n) (Laufzeit)

Während des Assemblierens eine Konstante, welche den Einsprung für PUSH enthält.
Während der Laufzeit wird ein Element auf dem Stapel abgelegt.

PUT (—a) (assemblieren)
(n—n') (Laufzeit)

Während des Assemblierens eine Konstante, welche den Ein sprung für den Einsprung PUT enthält.
Während der Laufzeit wird das oberste Element des Stapels über schrieben.

RP) Während des Assemblierens wird eine indirekt indizierte mit (101,X) erzeugt. Damit kann über Prozessor Stapelzeiger während der Laufzeit auf das oberste Element des RETURN-Stapels zugegriffen werden.

SEC (—A) (assemblieren)

Relative Adresse des niederwertigen Bytes des zweiten Elements auf dem Stapel. Wird beim Assemblieren in die indizierte indirekte Adressierung (2,X) gewandelt.

THEN, (A 2) (assemblieren)

Während des Assemblierens wird die Adresse A für noch nicht aufgelöste Sprünge benötigt. Die Zwei wird zur Fehlererkennung benutzt.

UNTIL, (A 1 cc) (assemblieren)

Während des Assemblierens wird ein bedingter Sprung, der von cc abhängt zur Adresse a assembliert. Die Eins dient zur Fehlererkennung.

UP (—A) (assemblieren)
Enthält Anfangsadresse der Benutzer Variablen.

W (—a) (assemblieren)
Adresse A enthält die Codefeldadresse des Wortes, das soeben ausgeführt wird.

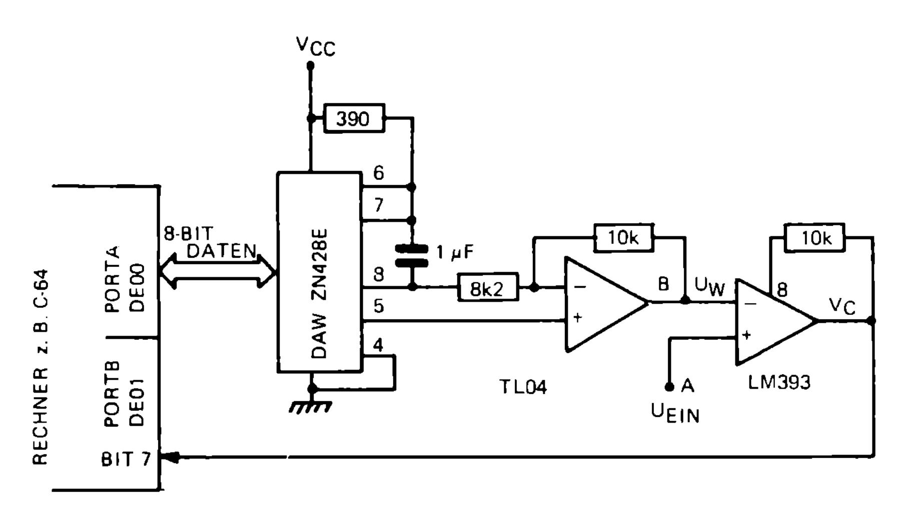
X) indiziert indirekte Adressierung

XSAVE (—a) (assemblieren)
Adresse einer Zelle in der Zero-Page zur Zwischenspeicherung des X-Registers.

Notizen

12 Analog-/Digital-Wandlung mit einem Digital-/Analog-Wandler

Die Abbildung 12.1 zeigt die Verwendung eines Digital-/Analog-Wandlers zur Umwandlung einer unbekannten Analog Spannung in eine digitale Zahl. Dabei wird das Verfahren der sukzessiven Approximation angewendet. Dieses Verfahren entspricht dem binären Suchen in sortierten Listen. Die unbekannte Spannung wird mit der Hälfte der maximalen Ausgangsspannung des Wandlers (UFS/2) verglichen. Ist die Eingangsspannung größer, so wird UFS/4 addiert, ist sie kleiner, so wird UFS/4 subtrahiert.



12.1 Analog-/Digital-Wandlung

Dieser Vorgang wird durch einen Rechner über Software gesteuert. In Abbildung 12.1 wird der Digital Analog Wandler ZN428E von einem C-64 über das Tor A mit der Adresse DE00 angesteuert. Der Analog Komparator LM393 vergleicht die Ausgangsspannung des Wandlers mit der unbekannten Eingangsspannung. Das Ausgangssignal des Komparators wird über Bit 7 von Tor B vom Rechner abgefragt. Das Ergebnis dieser Abfrage dient zur Verzweigung des Programms. Das Programm zeigt Abbildung 12.2.

```
SCR # 140
  0 ( FAW ADW MIT DAW       29.11.EF)
  1 HEX DE00 CONSTANT PORTA
  2 PORTA 1+ CONSTANT PORTB
  3 PORTA 2+ CONSTANT DDRA
  4 : INIT FF DDRA C! ;
  5 CODE CONV 80 # LDA, N STA,
  6 7F # LDA,
  7 BEGIN, DROP PORTA STA, NOP, NOP,
  8  PORTB LDY, 0< NOT
  9   IF, N ORA, THEN,
 10  N LSR, CS NOT
 11   IF, N EOR, ROT JMP,
 12 THEN,
 13 DEX, DEX, BOT STA, 0 # LDA,
 14 BOT 1+ STA, NEXT JMP, END-CODE
 15 ;S
```

12.2 Programm zur Wandlung einer Spannung in eine Zahl mittels sukzessiver Approximation

Die Adressen der Tore werden als Konstante vereinbart. Durch INIT wird Tor A zum Ausgang gemacht. Die Umwandlung wird durch das in Maschinensprache geschriebene Wort CONV ausgeführt. In der Hilfszelle N wird das Bitmuster %10000000 gespeichert. An den Wandler wird das Bitmuster %01111111 ausgegeben. Damit liegt an B die Spannung UFS/2. Diese Spannung wird mit der Eingangsspannung vergleichen. Ist UE kleiner als UW, dann ist UC = H. Das Tor B wird über das Y Register in den Rechner eingelesen. Da Bit 7 gleich Eins ist, ist dies eine negative Zahl. Dadurch wird mit 0 < NOT der Befehl N ORA, übersprungen. Der Inhalt der Speicherzelle N wird eine Stelle

nach links geschoben. Falls das CARRY-Bit dabei nicht gesetzt wird, muß mit der Wandlung fortgefahren werden. Dabei wird die oberste Eins im Akkumulator zu Null gemacht.

Ist also die Spannung UE kleiner als die Spannung UW, dann ist der Inhalt des Akkumulators

%00111111=$3F

nach dem ersten Vergleich. Dieser Wert wird an den Wandler ausgegeben und entspricht der Spannung UFS/4. Zwischen PORTA STA, und PORTB LDY, sind zwei NOP, NOP, Befehle eingefügt. Diese verzögern das Programm, um dem Komparator Zeit zum Einschwingen zu geben.

Ist nun die Eingangsspannung UE größer als UFS/4, so liegt an Bit 7 von Tor B eine Null. Damit wird nach 0 < NOT der zwischen IF, und THEN, stehende Befehl N ORA, ausgeführt. Damit wird eine Eins in das entsprechende Bit im Akkumulator übernommen. Damit ändern sich die Zelleninhalte wie folgt:

Vor Beginn des Vergleichs ist

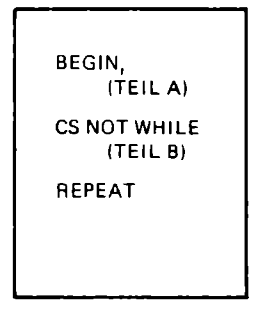
A=%00111111
N=%01000000

Nach dem Vergleich und nach der Befehlsfolge N ORA, N LSR, N EOR, ist

A=%01011111
N=%00100000 .

Ist Bit 7 gleich Null, so bleibt die Eins im Akkumulator erhalten, ist Bit 7 gleich Eins, so wird sie gelöscht.

Um dieses Programm mit dem Assembler schreiben zu können, musste die Prüfung des Assemblers auf exakte Strukturierung überlistet werden. Die Struktur des Programms ist in Abbildung 12.3 a angegeben.



12.3a Struktur des Wandelprogramms

Diese Befehle stehen im Assembler nicht zur Verfügung. Damit muß nach Abbildung 12.3b das Programm folgendermaßen geschrieben werden.

```
BEGIN,( -a1)
DROP
      (TEIL A)
CS NOT
  IF ( -b2)
      (TEIL B)
ROT JMP,
THEN,
```

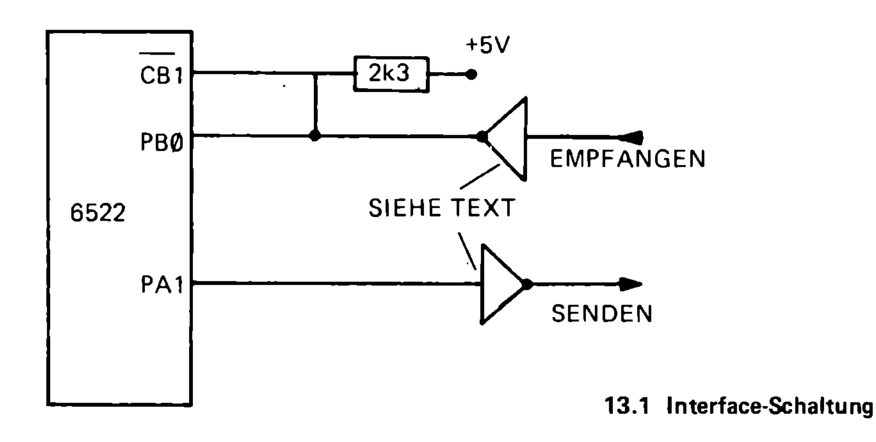
12.3b Geänderte Struktur

Das Wort BEGIN, legt die Rücksprungadresse und die Prüfzahl 1 auf dem Stapel ab. Das IF, nach der Abfrage legt seine Rücksprungadresse und die Prüfzahl 2 auf dem Stapel ab. Ist die Bedingung CS NOT erfüllt, so wird durch ROT die von BEGIN, auf den Stapel gelegte Rücksprungadresse in den TOS geholt und mit JMP, dorthin zurückgesprungen. Das auf JMP, folgende THEN, löst die Struktur der Verschachtelung auf.

13 Rechnerkopplung über RS232

Im folgenden wird ein Programm beschrieben, mit welchem Daten, Programme zwischen Rechner ausgetauscht werden können. Diese Rechner sind über eine 3-Drahtleitung verbunden. Eine Leitung wird für das Senden von Daten, die zweite Leitung für das Empfangen von Daten verwendet. Die dritte Leitung ist die gemeinsame Masse.

In dem Programm werden FORTH-Worte, gemischt mit Maschinenprogrammen verwendet. Es ist speziell für den APPLE II geschrieben, kann aber leicht auf andere Systeme übertragen werden. Die Abbildung 13.1 zeigt die Interface-Schaltung. Die Empfangsleitung ist mit dem Eingang PB0 und $\overline{CB1}$ angeschlossen. Eine negative Flanke an diesem Eingang löst einen Interrupt aus. Zwischen Eingang und dem 6522 Baustein muß unter Umständen noch ein Inverter oder ein RS232→TTL Wandler MC1489 geschaltet werden. Über PA1 werden die Daten ausgesendet. Auch hier muß meist ein Inverter oder TTL-RS232 Wandler MC 1499 nachgeschaltet werden.



13.1 Interface-Schaltung

13.1 Empfangen von Daten

Die Aufnahme von Zeichen erfolgt durch Interrupt. Damit ist sicher gestellt, daß Daten, die zum Beispiel über ein Modem ankommen, auch vom Rechner sofort erkannt und aufgenommen werden.

Das Wort IRQ löscht mit PORTB C@ das Interrupt-Flag Bit und gibt diesen mit 90 IER C! frei.

Wird nach IRQ eine negative Flanke an CB1 erkannt, so springt der Rechner, ganz gleich, in welcher Sprache er sich befindet, in die Inter rupt Routine IRQR. Hier werden zuerst die Register gerettet und dann das Unterprogramm HTIME durchlaufen. Damit wird der Abtastzeit punkt für die seriellen Daten in die Mitte des Bit-Impulses gelegt. Nach dem Startbit wird das erste Bit eingelesen. Ist es Eins, so wird das CARRY-Bit gesetzt und in die Zelle AUX geschoben. Nach jeder Ab tastung wird der Inhalt von AUX einmal nach rechts geschoben. Sind alle 7 Zeichen eingelesen, wird noch einmal mit LSR nach rechts geschoben. Damit ist das Zeichen vollständig übernommen. Nach der Freigabe des Interrupts und Übernahme der alten Registerwerte, wird über RTI in das unterbrochene Programm zurückgesprungen. Beim APPLE II wird der Inhalt des Accumulators beim Einsprung in die Interrupt-Service-Routine des APPLE II in der Zelle ACC gespeichert.

```
C400          26          ORG  $C400
C400          27    ;
C400 8A       28    IRQR  TXA
C401 48       29          PHA
C402 98       30          TYA
C403 48       31          PHA
C404 A900     32          LDA  #00
C406 A207     33          LDX  #07
C408 8DFEC4   34          STA  AUX
C40B 2052C4   35          JSR  HTIME
C40E 203EC4   36    IRQ0  JSR  TIME
C411 ADC0C0   37          LDA  PORTB
C414 2901     38          AND  #01
C416 38       39          SEC
C417 D001     40          BNE  IRQ2
C419 18       41          CLC
C41A 6EFEC4   42    IRQ2  ROR  AUX
```

```
C41D CA         43             DEX
C41E D0EE       44             BNE IRQ0
C420 4EFEC4     45             LSR AUX
C423 297F       46             AND #$7F
C425 ADFEC4     47             LDA AUX
C428 81FD       48             STA (BUF,X)
C42A E6FD       49             INC BUF
C42C D002       50             BNE IRQ3
C42E E6FE       51             INC BUF+1
C430 203EC4     52   IRQ3      JSR TIME
C433 ADC0C0     53             LDA PORTB
C436 58         54             CLI
C437 68         55             PLA
C438 A8         56             TAY
C439 68         57             PLA
C43A AA         58             TAX
C43B A545       59             LDA ACC
C43D 40         60             RTI
C43E            61   ;
C43E            62   ;
C43E ADBBC4     63   TIME      LDA TAB
C441 8DC4C0     64             STA T1CL
C444 ADBCC4     65             LDA TAB+1
C447 8DC5C0     66             STA T1CH
C44A ADCDC0     67   LT        LDA IFR
C44D 2940       68             AND #$40
C44F F0F9       69             BEQ LT
C451 60         70             RTS
C452            71   ;
C452 ADBBC4     72   HTIME     LDA TAB
C455 8DFEC4     73             STA AUX
C458 ADBCC4     74             LDA TAB+1
C45B 8DFFC4     75             STA AUX+1
C45E 4EFFC4     76             LSR AUX+1
C461 6EFEC4     77             ROR AUX
C464 ADFEC4     78             LDA AUX
C467 8DC4C0     79             STA T1CL
C46A ADFFC4     80             LDA AUX+1
C46D 8DC5C0     81             STA T1CH
C470 4C4AC4     82             JMP LT
C473            83   ;
```

13.2 Interrupt-Request Routine

Die Abbildung 13.2 zeigt die Interrupt-Service Routine. Diese wird in IRQON über den IRQ-Vektor in die Interrupt-Behandlung des Rechners eingebunden.

Das Wort INIT setzt das Datenrichtungsregister und liest die Maschinenprogramme aus Block 299 nach $C400.

Das Tor A wird als Ausgang programmiert. Die Zelle STATUS enthält den augenblicklichen Zustand des Tores A. In dieser Zelle wird bei einer Ausgabe über das Tor nur das Bit geändert, das ausgegeben wird. Damit ist sichergestellt, daß, wenn mehrere Geräte angeschlossen sind, nur an diese Daten ausgegeben wird. Als Ein- und Ausgabepuffer wird der Speicherbereich ab $4000 (Inhalt der Zelle BUF) verwendet. Das Hilfsregister ACR wird für die Betriebsart, Interrupt von Timer 1' und das Register PCR wird für CB1, Interrupt bei negativer Flanke an CB1' programmiert. Die Anfangsadresse der Interrupt-Service-Routine wird in den IRQ Vector $3FE (APPLE II) geschrieben.

Das Unterprogramm TIME holt die Verzögerungszeit für die gegebene Baudrate aus der Tabelle TAB, speichert sie in dem Zähler von Timer 1 und startet diesen. Dann wartet es in einer Schleife auf den Nulldurchgang des Timers. Dieser wird in Bit 6 des Registers IFR angezeigt.
Die programmierte Baudrate entspricht 300 Baud.
Das Unterprogramm HTIME verzögert nur die Hälfte der in der Tabelle programmierten Zeit. Die Division durch Zwei wird durch Linksschieben um eine Stelle durchgeführt.

Nach dem Aufruf der Worte INIT und IRQ ist der Rechner für die Aufnahme von Daten bereit.

Bei der Übertragung von Programmen wurden immer 2 Textfelder gleichzeitig übertragen. Hatte der sendende Rechner die Übertragung beendet, so wurde das im Speicher abgelegte Programm mit GET auf die Diskette übertragen. An GET wird die Nummer des Textfeldes übergeben, in welchen das Programm gespeichert wird.

13.2 Senden von Daten

Für die Ausgabe von Daten wird das Unterprogramm OUTCH in die Ausgaberoutine des APPLE II über den Vektor CSWL eingebunden.

Das Unterprogramm OUTCH1 sendet ein Zeichen mit Startbit, 7 Datenbit und zwei Stopbits. Für die Ausgabe einer Eins an PA0 wird die Zelle STATUS geholt und mit AND ≠$FE der augenblickliche Zustand der anderen Leitungen übernommen. Dann wird Bit 0 durch EOR ≠01 auf Eins gesetzt. Auf die gleiche Weise werden die Datenbits erzeugt. Die Zelle AUX wird eine Stelle nach links geschoben. Damit gelangt Bit Null in das CARRY-Bit. Ist das CARRY-Bit Eins, so wird eine Eins, sonst eine Null ausgegeben. Nach dem Datenbit folgen zwei Stopbits.

```
C473 8DFFC4    84  OUTCH1 STA AUX+1
C476 297F      85         AND #$7F
C476 49FF      86         EOR #$FF
C47A 8DFEC4    87         STA AUX
C47D ADFDC4    88         LDA STATUS
C480 29FE      89         AND #$FE
C482 4901      90         EOR #01
C484 8DC1C0    91         STA PORTA              ;STARTBIT
C487 203EC4    92         JSR TIME
C48A A207      93         LDX #07
C48C 4EFEC4    94  OUTO   LSR AUX
C48F ADFDC4    95         LDA STATUS
C492 29FE      96         AND #$FE
C494 9002      97         BCC 01
C496 4901      98         EOR #01
C498 8DC1C0    99  01     STA PORTA              ;DATENBIT
C49B 203EC4   100         JSR TIME
C49E CA       101         DEX
C49F D0EB     102         BNE OUTO
C4A1 ADFDC4   103         LDA STATUS
C4A4 29FE     104         AND #$FE
C4A6 8DC1C0   105         STA PORTA              ;STOPBIT
C4A9 203EC4   106         JSR TIME
C4AC ADFDC4   107         LDA STATUS             ;STOPBIT
C4AF 8DC1C0   108         STA PORTA
C4B2 203EC4   109         JSR TIME
C4B5 ADFFC4   110         LDA AUX+1
C4B8 4CF0FD   111         JMP $FDF0
```

```
C4BB         112  ;
C4BB 51      113  TAB    BYT $51
C4BC 0D      114         BYT $0D
C4C0         115         ORG $C4C0
C4C0 58      116         CLI
C4C1 60      117  .      RTS
C4C2         118  ;
C4C2         119  ;
            120         END
```

13.3 Unterprogramm OUTCH

Das Wort DLOAD holt den Inhalt von zwei Textfeldern und speichert diesen ab PAD. Das Wort SEND sendet diese Daten an das empfangende Gerät.

Als letztes Zeichen wird 01 gesendet. Dieses Zeichen kann beim empfangenden Rechner als Schlußzeichen ausgewertet werden.

```
SCR # 150
  0 ( RS232                    190983 EF)
  1 HEX  C0 CONSTANT SLOT
  2 C000 SLOT + CONSTANT PORTB
  3 C001 SLOT + CONSTANT PORTA
  4 C002 SLOT + CONSTANT DDRB
  5 C003 SLOT + CONSTANT DDRA
  6 C004 SLOT + CONSTANT T1CL
  7 C005 SLOT + CONSTANT T1CH
  8 C00B SLOT + CONSTANT ACR
  9 C00C SLOT + CONSTANT PCR
 10 C00D SLOT + CONSTANT IFR
 11 C00E SLOT + CONSTANT IER
 12 C4FD CONSTANT STATUS
 13   FD CONSTANT BUF
 14   36 CONSTANT CSWL
 15  3FE CONSTANT IRQVEC  -->
```

```
SCR # 151
  0 ( RS232 CNTD                       EF)
  1 : SL ( -N) 4 ;
  2 : INIT FF DDRA C! 02 PORTA C!
  3   2 STATUS C! 0 ACR C! 0 PCR C!
  4   C400 12B 1 R/W ;
  5 : CSWLON 73 CSWL C! C0 SL +
  6   CSWL 1+ C! ;
  7 : CSWLOFF F0 CSWL C! FD
  8   CSWL 1+ C! ;
  9
 10 : IRQON 00 IRQVEC C! C0 SL +
 11   IRQVEC 1+ C! ;
 12 : IRQOFF 65 IRQVEC C! FF IRQVEC
 13   1+ C! ;
 14 DECIMAL    -->
 15
```

```
SCR # 152
  0 ( RS232 EMPFANGEN      190983 EF)
  1 HEX
  2 : IRQ 00 BUF C! 40 BUF 1+ C!
  3   PORTB C@ 90 IER C!
  4   IRQON C4C0 CALL ;
  5
  6
  7
  8 DECIMAL
  9
 10     -->
 11
 12
 13
 14
 15
```

```
SCR # 153
  0 ( RS232 EMPFANG                   EF)
  1 16384 CONSTANT C
  2 0 VARIABLE CC
  3
  4 : GET ( N) 2 * C CC ! 4 0 DO
  5   CC @ OVER I + 0 R/W 256 CC +!
  6   LOOP DROP ;
  7
  8      -->
  9
 10
 11
 12
 13
 14
 15
```

```
SCR # 154
  0 ( RS232 SENDEN                    EF)
  1
  2 : DLOAD ( N) 2 * DUP 5 + SWAP
  3   PAD CC ! DO
  4   CC @ I 1 R/W 256 CC +! LOOP ;
  5
  6 : (SEND) ( NN') CSWLON DO I C@
  7   EMIT LOOP 01 EMIT CSWLOFF ;
  8 : SEND ( N) DLOAD PAD DUP 1024
  9   + SWAP (SEND) ;
 10
 11
 12
 13
 14
 15
```

13.4 RS232-Übertragung

14 BUSIPACK

Das BUSIPACK enthält eine Lagerverwaltung, eine Adressverwaltung und ein Rechnungsschreibprogramm.

Die Adressverwaltung entspricht im wesentlichen der Adressverwaltung in Kapitel 9. Die Lagerverwaltung ist ähnlich aufgebaut. Beide benutzen die gleichen Worte zum Speichern der Daten auf Diskette. Durch das Wort C( )C werden die Konstanten START und RECLEN ausgetauscht. Die Lagerdatei beginnt bei Block 320 und hat eine Datensatzlänge von 64 Bytes. Die Adressverwaltung beginnt bei Block 80 und ein Datensatz ist 128 Bytes lang.

Die Blöcke 80, 318 und 319 enthalten die folgenden Eingaben:

```
SCR # 40
  0                % RABATT -
  1      └──── 13 Leerzeichen
  2

SCR # 159
  0  TELEFON          TELEX
  1              ↑
  2              └── 7 Leerzeichen
  3
  4
  5
  6
  7
  8
  9  ZUR ZEIT VERGRIFFEN
 10  WIRD VORGEMERKT
 11  KEINE GUELTIGE BESTELLNUMMER
```

Es würde zu umfangreich werden, hier alle Worte zu erklären. Bei der Implementierung dieses Programms sollte nach folgenden Schritten vorgegangen werden.

Zuerst wird die Adressliste programmiert. Die hier angegebene Liste ist mit der in Kapitel 9 beschriebenen austauschbar. Danach sollte die Lagerverwaltung programmiert werden. Diese ist nach ähnlichen Gesichtspunkten aufgebaut. Wenn diese beiden Teile fertig sind, dann kann das Programm zum schreiben einer Rechnung programmiert werden.

BUSI-PACK

Das Programm BUSI-PACK vereinigt die beiden Programme SUPER MAIL und SUPERINVENTORY zu einem Programm, mit welchem Rechnungen geschrieben werden können. Gleichzeitig wird das Lager verwaltet. Die in den Rechnungen aufgeführten Artikel werden automatisch aus dem Lagerbestand ausgetragen.

Die Bestellungen werden in einen Bestell-File eingetragen. Im Anschluß an die Eingabe werden die Rechnungen geschrieben. Auf einer Diskette können die folgenden Daten aufgezeichnet werden:

- Ein Bestell-File kann bis zu 40 Bestellungen aufnehmen.
- Im Lagerprogramm können 900 Artikel gespeichert werden.
- Die Adressverwaltung kann 700 Adressen aufnehmen.

Die Bedienung der Programmteile ADRESSEN und LAGER entspricht den Programmen SUPERMAIL und SUPERINVENTORY. Datendisketten, die mit diesen Programmen beschrieben worden sind, können jedoch nicht mit dem BUSI-PACK verwendet werden, da die Datenverteilung auf der Diskette unterschiedlich ist.

Erste Inbetriebnahme des BUSI-PACKS

Zuerst werden nach dem üblichen Verfahren, ein oder zwei Disketten formatiert. Dies sind die späteren Datendisketten. Dann wird die Programm-Diskette in das Laufwerk eingelegt und der Rechner eingeschaltet. Mit LOAD "*",8 wird das Programm geladen und mit RUN gestartet.

Die formatierten Disketten müssen noch initialisiert werden. Dies geschieht durch das Wort ALLNEW. Nach der Eingabe des Wortes wird die Programmdiskette aus dem Laufwerk genommen und die DATA INIT Diskette in das Laufwerk eingelegt. Diese Diskette ist die Rückseite der Programmdiskette. Nach Betätigen der RETURN-Taste wird auf Anforderung die formatierte Diskette eingelegt und durch ein weiteres RETURN initialisiert. Danach erscheint das Menue des Adressenprogramms. Soll eine zweite Diskette initialisiert werden, so wird auf die Anfrage WAS wiederum ALLNEW eingegeben.

Zwischen den einzelnen Programmteilen wird jeweils mit LAGER RUN bzw. ADRESSEN RUN umgeschaltet. Wird also im Menue des Adressenprogramms LAGER RUN eingegeben so wird das Lagerprogramm aktiv. Umgekehrt wird mit der Eingabe von ADRESSEN RUN im Lagerprogramm das Adressenprogramm aktiviert.

Auf eine neue Diskette müssen die Firmenanschrift und die Bankverbindungen eingegeben werden. Die Firmenanschrift ist jeweils die erste Adresse in der Adressverwaltung. Die Bankverbindungen sind in den Feldern VN und CO der beiden nächsten Adresseintragungen angegeben. Für die Eingabe wird mit

ADRESSEN RUN

in das Adressprogramm gesprungen und dort mit

TERMINAL EINGABE

die Eingabe der Adresse begonnen. Die Adressmaske erscheint auf dem Bildschirm, mit einer Eins links oberhalb der Maske. Ein Beispiel zeigt die Abbildung (Eingabe einer Firmenanschrift) auf der nächsten Seite.

```
-----------------------------------
-VERLAG DER COMPUTERFREUNDE  -- ---
-----------------------------------
-                       ----------
-----------------------------------
-GRUENE BAUMSTR. 32     ----------
-----------------------------------
-D-8000 -MUENCHEN             -----
-----------------------------------
-     -1234567-089/1234567------  -
-----------------------------------
WAS
```

Eingabe einer Firmenanschrift

In das Codefeld C2 wird die TELEX Nummer, in das Feld TEL die Telefonnummer eingetragen.

In die nächsten beiden Adressen werden die Bankverbindungen oder ein anderer Text in die Felder VN und CO eingetragen. Von 46 möglichen Zeichen werden 45 als laufender Text ausgegeben. Ein Beispiel für die Eingabe zeigt die nächste Abbildung. Sollen keine Bankverbindungen gedruckt werden, so müssen in die Adresseinträge 2 und 3 Leereingaben gemacht werden.

```
-----------------------------------
-DRESDNER BANK MUENCHEN 5 390------
-----------------------------------
- 370 BLZ 700 800 00    ----------
-----------------------------------
-                       ----------
-----------------------------------
-       -                     -----
-----------------------------------
-    -       -            -----  -
-----------------------------------
WAS
```

Eingabe der Bankverbindungen

Sind diese Eingaben gemacht worden, so ist die Adresskartei zur Aufnahme von Adressen bereit. Diese können direkt oder über das Rechnungsschreibprogramm eingegeben werden.

Als nächstes werden die Artikel in das Lagerverzeichnis eingegeben.

ADRESSVERWALTUNG

Die Adressverwaltung wird im BUSI-PACK mit ADRESSEN RUN gestartet. Danach erscheint das Hauptmenue:

```
                 ADRESS-VERWALTUNG

      TERMINAL EINGABE
               SUCHE    &lt;BZ&gt; &lt;NAME&gt;
          &lt;NR&gt; FINDE
               INHALT


      DRUCKER  LABEL
               SUCHE    &lt;BZ&gt; &lt;NAME&gt;
               INHALT
          &lt;NR&gt; DRUCKE

      RUN
      NEU
      RECHNUNG
      WAS
```

Menue Adressverwaltung

Eingabe von Adressen:

Das Wort TERMINAL braucht nur zu Anfang oder nach dem Ausdrucken von Adressen eingegeben zu werden. Nach dem Wort EINGABE erscheint die Eingabemaske:

```
 2
------------------------------------
-VN                            -----
------------------------------------
-CO                       ----------
------------------------------------
-ST                       ----------
------------------------------------
-PLZ     -ORT                  -----
------------------------------------
-C1    -C2      -TEL          ----- -
------------------------------------
WAS
```

Eingabemaske

Über der Maske wird die laufende Nummer des Eintrags angezeigt. Diese Nummer wird auch in der Liste aund auf den Etiketten ausgedruckt. Die Bezeichnungen der einzelnen Felder sind in der Abbildung angegeben. Sie erscheinen nicht in der Maske. In das Feld VN wird der Name eingetragen. Eine Zusatzangabe kann in dem Feld CO gemacht werden. Der Übergang von einem Feld in das andere erfolgt durch die RETURN-Taste oder automatisch, wenn das Feld voll beschrieben ist. In die Felder ST, PLZ und ORT werden die Straße, die Postleitzahl und der Ort eingetragen.

Als Korrekturtaste kann nur die INST/DEL-Taste verwendet werden. Alle anderen Cursortasten sind wirkungslos.

Die Felder C1 und C2 sind Codefelder. Hier können Bezeichnungen eingetragen werden, mit denen später Adressen selektiert werden können.

Beispiel:
Alle ATARI-Besitzer haben in der ersten Stelle des Codefeldes ein A, alle APPLE-Besitzer ein B und alle ZX81-Besitzer ein Z. Mit

DRUCKER SUCHE C1 A

werden Adressaufkleber nur für ATARI-Besitzer und mit

DRUCKER SUCHE C1 B

nur für APPLE-Besitzer ausgedruckt. Es empfiehlt sich, diese Codebezeichnungen vor dem Anlegen einer Adressdatei sich genau zu überlegen.

In das letzte Feld wird die Telefonnummer eingetragen. Diese letzte Zeile wird bei den Aufklebern nicht ausgedruckt. Sind alle Eingaben gemacht worden, so wird durch OK (J/N) angefragt, ob die Eingabe richtig ist. Zur Bestätigung kann die J oder die RETURN-Taste gedrückt werden. Bei einer Verneinung mit N wird der Eintrag nicht gespeichert. Wird die Anfrage MEHR (J/N) mit J oder RETURN beantwortet, so können weitere Einträge gemacht werden. Mit N wird in das Hauptmenue zurückgesprungen.

Suchen von Adressen

Mit der Eingabe

SUCHE ⟨BZ⟩ ⟨NAME⟩

kann nach einer Adresse gescuht werden. ⟨BZ⟩ ist die Bezeichnung des Feldes, ⟨NAME⟩ der Eintrag, nach welchem gesucht wird. Die Eingabe von:

SUCHE PLZ D–8

sucht alle Adressen aus, deren Postleitzahl mit D–8 beginnt.

Die Eingabe von

SUCHE VN FRANK MUELLER

sucht alle Adressen aus, die den Vornamen FRANK haben. Der Vergleich der Zeichenketten wird nur bis zu einem Leerzeichen durchgeführt. Ist vor SUCHE das Wort TERMINAL eingegeben worden, so werden immer drei gefundene Adressen auf dem Bildschirm angezeigt. Wird die Leertaste gedrückt, so wird die Suche fortgesetzt. Mit der

Taste X wird die Suche abgebrochen. Ist vor dem Wort SUCHE das Wort DRUCKER eingegeben worden, so werden Adressenaufkleber aus gedruckt.

Auffinden von Adressen

Jede Adresse erhält eine fortlaufende Nummer, die immer auf dem Bildschirm angezeigt und ausgedruckt wird. Mit

5 FINDE

wird der fünfte Eintrag auf den Bildschirm angezeigt. Nun erscheint am oberen Bildrand eine neue Menue-Zeile.

AENDERE ⟨BZ⟩ LOESCHEN EINTRAG RUN

Mit dem Wort AENDERE können die Einträge in den einzelnen Feldern geändert werden. Die Eingabe

AENDERE CO

ändert die Eintragung in das Feld CO. Der Cursor sitzt unter dem Wort WAS und ein ⟨ zeigt die Länge der möglichen Eingabe an. Gibt man auf die Anfrage OK (J/N) J oder RETURN ein, so wird der geänderte Eintrag auf die Diskette zurückgeschrieben. Wird mit N geantwortet, so können weitere Änderungen gemacht werden.

Das Wort LOESCHEN löscht diese Adresse. An diese Stelle kann mit EINTRAG eine neue Adresse eingegeben werden. Das Wort RUN bringt das Menue wieder auf den Bildschirm.

Das Wort INHALT gibt alle Adressen entweder auf den Bildschirm oder auf den Drucker aus, je nachdem TERMINAL oder DRUCKER eingegeben worden ist.

Adressaufkleber von der gesamten Adressdatei werden durch DRUCKER LABEL zweispaltig ausgedruckt. Sind an eine Adressdatei neue Adressen angehängt worden, so können Adressaufkleber von diesen Adressen durch das Wort DRUCKE ausgedruckt werden.

Die Eingabe

135 DRUCKE

druckt ab Adresse 135 bis zum Ende der Liste aus.

Man beachte, vor FINDE und DRUCKE muß immer eine Zahlenangabe sein.

Sollen auf einer Datendiskette die Adressen gelöscht werden, so wird

ADRESSEN NEU

eingegeben. Damit sind alle Adressen gelöscht und in die Adresskartei können neue Eingaben gemacht werden.

LAGERVERWALTUNG

Das Programm Lagerverwaltung dient zur Erfassung und Verwaltung eines Lagerbestandes. Auf einer Diskette können bis zu 900 Artikel erfasst werden. Die Länge eines Datensatzes beträgt 64 Byte.

Für einen Artikel wird eine lagernummer (4-stellig), eine Bezeichnung (20 Stellen), ein Codefeld für die Mehrwertsteuer und ein Hersteller-schlüssel (2 Stellen) eingegeben. Weitere Einträge sind die Mindest-menge, die im Lager vorhanden sein muß und der Verkaufspreis.

Danach folgen die Lagermenge, die verkaufte Menge und das Datum des letzten Zugriffs auf den Artikel. Die beiden letztgenannten Einträge werden vom Programm aus vorgenommen. Sie werden bei der Eingabe übersprungen. Die letzte Eintragung ist der Einkaufspreis.

Im BUSI-PACK wird das Programm mit LAGER RUN gestartet. Es erscheint das Hauptmenue des Programms:

```
             LAGERVERWALTUNG

            RUN
            DATUM TTMMYY
            EINGABE
            ABGANG
            ZUGANG
            DRUCKER
            <#> AUSGABE
            NEU
            FIN
            INHALT
            BACKUP
      WAS
```

Menue Lagerprogramm

Der Befehl NEU löscht eine vorhandene Lagerdatei. Deshalb erscheint nach NEU die Frage

WIRKLICH? (J/N)

Wird diese Frage mit J beantwortet, so ist die Lagerdatei auf der Diskette gelöscht. Mit N wird der Befehl NEU aufgehoben. Der Befehl DATUM wird zur Eingabe des gültigen Datums verwendet. Die Eingabe erfolgt in der Form

DATUM 020983

für den 2.9.1983. Zwischen dem Wort DATUM und der Zahlenreihe ist nur ein Leerzeichen. Diese Eingabe kann mehrmals hintereinander erfolgen.

Der Befehl RUN initialisiert nur die Maske. Er hat keine Auswirkungen auf den Programmablauf. Bei Fehleingaben erscheint ein ?. Danach kann der Befehl neu eingegeben oder die Maske mit RUN gelöscht und danach neu eingegeben werden. Bei Tippfehlern kann mit der INST DEL Taste ein oder mehrere Buchstaben gelöscht werden. Nur diese Taste kann zur Korrektur verwendet werden. Alle anderen Tasten sind dafür nicht verwendbar.

Die Eingabe von Lagerbeständen wird durch den Befehl EINGABE eingeleitet. Es erscheint die Eingabemaske.

Unter der Eingabemaske ist eine neue Befehlsmaske mit abgekürzten Befehlen zu sehen. Wird auf die Frage WAS der Buchstabe E für E)INGABE eingegeben, so wird der Cursor des Bildschirms auf die erste Stelle der Lagernummer gesetzt. Die Punkte hinter den Bezeichnungen geben die Zahl der Stellen an, die eingegeben werden können. In das nächste Eingabefeld wird nach der Eingabe aller Stellen oder nach RETURN übergegangen.

```
 DATUM: 4.04.84


LAGER#          : ....
BEZEICHNUNG     : ....................
MWST CODE       : .
HERSTELLER      : ..
MINDEST MENGE   : ...
VERKAUFSPREIS   : ........
LAGERMENGE      : .....
VERK. MENGE     : .....
LETZTER ZUGRIFF: ....
EINKAUFSPREIS   : ........


 WAS

--------------------------------------
E)INGABE EN)DE ME)HR EX)IT SP)EICHERN
LA BE MW HE MM VKP LAM VKM LD EKP
```

Eingabemaske Lager

Die Lagernummer muß eine Zahl mit maximal 4 Stellen sein. Eingaben wie 010, 4135 oder 12 sind erlaubt, nicht dagegen 1C5 oder A123. Das nächste Feld enthält die Bezeichnung mit maximal 20 Stellen.

Für das Schreiben einer Rechnung wird ein Codeschlüssel für die Mehrwertsteuer benötigt. Dieser kann in das nächste Feld eingegeben

werden. Eine 1 bezeichnet die ermäßigte, eine 2 die volle Mehrwertsteuer. Es folgt eine zweistellige Angabe für den Hersteller, die Mindestmenge und der Verkaufspreis. Dieser muß mit Dezimalpunkt eingegeben werden (z. B. 19.50).

Nach der Eingabe der Lagermenge überspringt der Cursor die beiden nächsten Felder und verlangt nach der Eingabe des Einkaufspreises. Soll in ein Feld keine Eintragung gemacht werden, so kann ein Leerzeichen eingegeben oder mit der RETURN Taste in das nächste Feld übergangen werden.

Sind alle Eingaben gemacht worden, so wartet das Programm auf einen neuen Befehl durch die Frage WAS.

Sind bei der Eingabe Fehler gemacht worden, so können die Worte in der zweiten Reihe verwendet werden, um Einträge in den einzelnen Feldern zu korrigieren. Das Wort LA zum Beispiel ändert die Lagernummer. Die Eingabe von

LA ⟨RETURN⟩

setzt den Cursor auf die erste Stelle der Lagernummer. Nun kann diese neu eingegeben werden. Nach der Eingabe erfolgt wieder die Frage WAS. Die Reihenfolge der Befehle in der zweiten Zeile entspricht der Reihenfolge der Felder. Diese können in beliebiger Reihenfolge und auch mehrfach hintereinander eingegeben werden.

Wenn weitere Einträge gemacht werden sollen, so wird durch ME für ME)HR der eben gemachte Eintrag gespeichert und eine neue Lagermaske auf den Bildschirm ausgegeben. Zur Beendigung der Eingabe wird EN für EN)DE eingegeben. Der letzte Eintrag wird gespeichert und das Programm springt in das Lagermenue zurück. Mit den Buchstaben EX für EX)IT wird ebenfalls in das Lagermenue zurückgesprungen. Dabei wird aber der letzte Eintrag nicht gespeichert.

Die Ausgabe eines Eintrags auf den Bildschirm erfolgt durch # AUSGABE. Durch

170 AUSGABE

wird der Inhalt des Datensatzes mit der Lagernummer 170 auf den Bildschirm ausgegeben. Zwischen der Zahl und dem Wort AUSGABE ist nur ein Leerzeichen.

Nun können Änderungen wie bei der Eingabe vorgenommen werden. Dieser Eintrag darf nun nicht mit EN oder ME in die Datei zurückgeschrieben werden. Damit würde er ans Ende der Datei und nicht an seinen richtigen Platz in der Datei zurückgeschrieben. Zur Speicherung wird der Befehl SP für SP)EICHERN verwendet. Dieses Wort speichert einen geänderten Eintrag in die Datei zurück. Ein Eintrag in die Datei kann nicht gelöscht, aber auf diese Weise überschrieben werden.

Für die Veränderung des Lagerbestandes werden zwei Befehle, ZUGANG und ABGANG verwendet. Nach

ABGANG ⟨RETURN⟩

wird nach der Lagernummer gefragt und der Eintrag angezeigt. Nach Eingabe der Menge wird diese in dem Datensatz geändert und das Datum eingetragen. Durch ZUGANG wird auf die gleiche Weise der Lagerbestand geändert.

Durch das Wort DRUCKER wird eine Lagerliste ausgedruckt. Die Ausgabe erfolgt in der Reihenfolge der Eingabe.

Zusatz für C-64:
Drucker an IEEE Bus angeschlossen.

Zusatz für ATARI:
850-Interface oder ELCOMP RS232 verwenden.

Zusatz für APPLE:
Printer Interface in Slot 1.

Das Wort INHALT gibt den Artikelbestand mit Lagernummer, Bezeichnung, Lagermenge und Verkaufspreis auf den Bildschirm aus.

Vor dem Ausschalten des Computers muß das Wort FIN eingegeben werden. Dadurch werden alle noch nicht auf der Diskette gespeicherten Einträge zurückgeschrieben.

Von dem Lagerbestand kann ein BACKUP gemacht werden. Dazu wird eine formatierte Diskette mit ALLNEW initialisiert. Dann wird die Originaldiskette in das Laufwerk eingelegt und

LAGER BACKUP

eingegeben. Nach der Aufforderung

NEUE DISKETTE EINLEGEN
RETURN DRUECKEN

wird die Originaldiskette aus dem Laufwerk genommen und die neue Diskette eingelegt. Nach einiger Zeit erscheint die Aufforderung

ORIGINAL DISKETTE EINLEGEN
RETURN DRUECKEN

Nun wird wieder die Originaldiskette eingelegt. Dieser Zyklus wiederholt sich noch vier Mal.

Achtung!
Beim Lesen der Diskette blinkt die Leuchtdiode so, als ob ein Diskettenfehler vorliegt. Die Datenblöcke werden in diesem Rhytmus eingelesen.

BUSI-PACK (Übersicht)

Das Programm BUSI-PACK besteht aus zwei Disketten, der Programm-Diskette und (auf der Rückseite) der DATA-INIT Diskette. Auf beiden Disketten darf auf keinen Fall der Schreibschutz entfernt werden. Sie dürfen auch nicht als Datendisketten verwendet werden, da sonst das Programm zerstört wird. Als Datendisketten werden formatierte Disketten verwendet.

Das Programm wird mit LOAD "*",8 aus BASIC geladen. Nach RUN erscheint das Startmenue.

```
BUSI-PACK

(C) 1984 BY ING W. HOFACKER GMBH

  ALLNEW
  ADRESSEN RUN
  LAGER RUN
DATEN DISKETTE EINLEGEN

WAS
```

Startmenue BUSI-PACK

Mit LAGER RUN wird das Lagerverwaltungs-Programm, mit ADRESSEN RUN das Adressenverwaltungsprogramm gestartet. Mit ALLNEW wird eine neue Datendiskette für das BUSI-PACK erstellt.

In das Rechnungsschreibprogramm gelangt man aus dem Menue des Adressenprogramms mit dem Wort RECHNUNG. Dann erscheint das Menue dieses Programmteils.

```
  RECHNUNGEN SCHREIBEN


DATUM :4.04.84
MWST  :14.0%
RECHN#:1000

BSTF NEUER BESTELL-FILE
CONT BESTELL-FILE ERGAENZEN
RESS RECHNUNGEN SCHREIBEN

WAS
```

Menue Rechnungsschreiben

Mit BSTF wird ein neuer Bestell-File eröffnet. Dessen Menue hat folgendes Aussehen:

```
WAS

-----------------------------------------
   ANSCHRIFT       BES)TELLUNG

   SON)DERARTIKEL WEITER

   ENDE
```

Menue Bestell-File

Die Eingabe einer Bestellung beginnt mit dem Wort ANSCHRIFT. Zuerst wird nach der Kundennummer gefragt. Dies ist die Nummer, unter welcher die Adresse des Kunden in der Adressverwaltung eingetragen ist. Ist diese bekannt, so wird nach der Eingabe der Nummer die Adresse auf dem Bildschirm angezeigt.

Mit J oder der RETURN-Taste wird in der Eingabe fortgefahren, mit N kann eine neue Nummer eingegeben werden.

Ist die Kundennummer nicht bekannt, so wird nur die RETURN-Taste gedrückt. Das Programm schaltet dann in das Adressenprogramm um und gibt die Adressmaske zur Eingabe der Adresse auf den Bildschirm aus. Ist die Adresse eingegeben worden, so wird in der Aufstellung der Anschrift fortgefahren. Die nächste Eingabe ist die Bestellnummer des Kunden. Diese kann 10 Stellen lang sein. Es folgen die Eingabe des Bestell-Datums und die Angabe, ob der Kunde Mehrwertsteuer bezahlt. Bei Bejahungen kann entweder die Taste J oder RETURN, bei Verneinungen muß die Taste N gedrückt werden.

Mit der nächsten Eingabe wird der Rabatt festgelegt. Hier gibt es drei Möglichkeiten. Wird mit N verneint, so wird zur nächsten Eingabe weitergegangen. Wird mit J geantwortet, so erscheint die Frage SONDERRABATT. Wird nun mit N geantwortet, so wird der Rabatt nach folgender Staffel berechnet.

Stückzahl > 10 40%
Stückzahl > 5 33%
Stückzahl >= 1 25%

Bei der Berechnung des Rabatts wird jeweils die Gesamtstückzahl berücksichtigt. Wird die Frage nach dem Sonderrabatt mit J beantwortet, so wird dieser danach als zweistellige Zahl eingegeben. Beispiel: 35 für 35%. Als letzte Eingabe erfolgt die Eingabe der Versandkosten. Werden keine Versandkosten berechnet, so wird die RETURN-Taste gedrückt. Andernfalls wird ein Betrag mit Dezimalpunkt eingegeben (6.00 für DM 6.00 Versandkosten).

```
BESTELL-NR    :1234567890
BESTELL DATUM:030484
STEUER ( /N) :J
RABATT ( /N) :J
SONDER-RABATT:A


VERSANDKOSTEN:5.00
```

Eingabe Anschrift

Nun ist die Eingabe der Anschrift beendet. Das Programm verlangt mit OK (J/N) eine Bestätigung. Mit J ist die Eingabe beendet, mit N beginnt sie aufs Neue.

Als nächstes erfolgt die Eingabe der Bestellungen. Hier gibt es zwei Möglichkeiten. Sind die Artikel im Lager eingetragen, so wird mit BEStellung weitergemacht. Es dürfen nur die Buchstaben BES eingegeben werden. Die einzelnen Bestellungen werden in folgendes Schema eingetragen:

```
BEST. MENGE :10
GEL.  MENGE :10
BESTELL NR  :1000
```

Eingabe Bestellung

Nach der Eingabe der Lager- oder Bestellnummer wird mit OK (J/N) und mit MEHR (J/N) abgefragt, ob die Eingaben richtig sind und ob weitere Eingaben gemacht werden sollen. Mit N wird die Eingabe ab-

geschlossen.

Die zweite Möglichkeit ist die Eingabe eines SONderartikels. Dies sind Artikel, die nicht in das Lagerverzeichnis eingetragen sind. Nach der Eingabe von SON erscheint die Eingabemaske.

```
10 ASCHENBECHER ....... 2 3.75 ..
```

Eingabe eines Sonderartikels

Die Anzahl ist maximal zweistellig und muß mit RETURN abgeschlossen werden. Dann folgt die Bezeichnung, der Mehrwertsteuerschlössel (1 = halbe MWST., 2 = volle MWST) und der Preis. Aus dem augenblicklichen Eingabefeld wird, wenn dieses nicht vollgeschrieben ist, in das nächste mit der RETURN-Taste gesprungen. Deshalb kein RETURN nach der Eingabe des MWST-Schlüssels. Mit OK (J/N) wird die Richtigkeit der Eingabe bestätigt. Wird die Frage MEHR (J/N) mit N beantwortet, so ist diese Eingabe abgeschlossen.

Eine Bestellung kann aus 15 Einzelposten bestehen, wobei Lagerartikel und Sonderartikel gemischt eingegeben werden können. Aus Platzgründen können aber nur maximal 7 Sonderartikel eingegeben werden.

Eine weitere Bestellung wird mit WEITER, gefolgt von ANSCHRIFT usw. in den Bestell-File aufgenommen. Mit ENDE wird der Bestell-File abgeschlossen.

Ein bereits bestehender Bestell-File kann mit dem Wort CONT erweitert werden. Dazuwischen können Änderungen im Lager oder bei den Adressen vorgenommen werden. CONT kann nicht mehr verwendet werden, wenn durch RESS Rechnungen geschrieben worden sind.

Das Menue des Rechnungs-Schreib-Programms zeigt oben das Rechnungsdatum, die Mehrwertsteuer und die nächste Rechnungsnummer. Diese Angaen können folgendermassen geändert werden:

DATUM TTMMJJ setzt das Datum.

Beispiel:
DATUM 101283 setzt das Datum auf den 10. Dezember 1983.

MWST NNN setzt die Mehrwertsteuer.

Beispiel:
MWST 140 ergibt die Mehrwertsteuer 14.0%.

RECHN# NNNN setzt die Rechnungsnummer.

Beispiel: RECHN# 2145 setzt die Nummer für die nächste Rechnung auf 2145.

Die Eingaben können in beliebiger Reihenfolge und mehrfach hintereinander gemacht wwerden.

In dem Wort RESS beginnt, nachdem das Papier eingelegt und dier Drucker eingeschaltet worden ist, der Ausdruck der Rechnung.

Die Stellung des Druckkopfes für den Ausdruck auf das Formularpapier muß ausprobiert werden. Beim Abschalten des Druckers nach dem Ausdrucken der letzten Rechnung wird noch ein Zeilenvorschub zum Leeren des Druckpuffers ausgegeben. Werden Rechnungen einzeln mit BSTF und RESS ausgedruckt, so muß das Formularpapier um einen Zeilenvorschub zurückgesetzt werden.

Einige allgemeine Bemerkungen zum Programm

Das Programm BUSI-PACK wird durch Worte gesteuert. Werden bei der Eingabe Schreibfehler gemacht, so antwortet das Programm mit dem eingegebenen Wort und Fragezeichen. Danach kann dieses Wort neu eingegeben werden. Das Wort RUN gibt in beiden Programmteilen nur die Maske des Menues auf den Bildschirm aus. Es kann deshalb, ohne daß dadurch am Programm oder den Daten geändert wird, auch zum Löschen und Neuerstellen des Menues verwendet werden. Die RESTORE Taste führt einen Warmstart durch. Allerdings führt das Drücken der RESTORE Taste während des Druckens unter Umständen zum Systemabsturz.

```
SCR # 110
  0 ( COMMON SCREENS BEGIN       EF)
  1  FORTH DEFINITIONS
  2  0 VARIABLE H 0 VARIABLE V
  3  : HO ( n) 0 DO SPACE 1 H +!
  4  LOOP ;
  5  : VE ( n) 0 DO CR   1 V +!
  6  0 H ! LOOP ;
  7 : ATYPE -DUP IF OVER + SWAP
  8   DO I C@ 127 AND DUP 0=
  9   IF DROP ELSE EMIT 1 H +!
 10  THEN  LOOP ELSE DROP ENDIF ;
 11 : PRINT ( NA) PAD + SWAP
 12   -TRAILING ATYPE ;
 13
 14      -->
 15

SCR # 111
  0  ( BUSI FORMAT                ef)
  1  : ADJ ( n) H @ - -DUP IF HO
  2    THEN ;
  3  : PTEX> ( nan') ADJ PRINT ;
  4  : TEX>  ( ann') ADJ -TRAILING
  5    ATYPE ;
  6  : (RADJ) ( ann'n") SWAP ADJ
  7    SWAP DUP >R - HO R> ATYPE ;
  8  : #N ( d)    <# #S #> ;
  9  : #DM ( d) <# # # 46 HOLD #S
 10    #> ;
 11  : #% ( d) <# 37 HOLD # 46 HOLD
 12    #S #> ;
 13  : #DA ( d) <# # # 46 HOLD # #
 14    46 HOLD #S #> ;
 15      -->

SCR # 112
  0  ( INPUT WORDS                ef)
  1  : PADD PAD 128 + ;
  2  : IP PAD + SWAP EXPECT ;
  3  : IPP PADD + SWAP EXPECT ;
  4  : PADDC PADD 128    32 FILL ;
  5  : ?IP PADDC OVER PADD SWAP
  6  EXPECT PADD C@ IF PADD SWAP
  7  PAD + ROT    CMOVE THEN ;  -->
  8
  9
 10
 11
 12
 13
 14
 15
```

```
SCR # 113
  0  ( INPUT WORDS                 ef)
  1  0 VARIABLE CNT
  2  : PADC PAD 128    32 FILL ;
  3 : ?Z ( N-NF) DUP 47 > OVER 58 <
  4   AND OVER 46 = OR ;
  5 : (S) 32 C, -1 ALLOT ;
  6 : (Z> ( -D) CNT @ HERE OVER -
  7   SWAP 0= NOT IF NUMBER ELSE
  8   DROP THEN ;
  9 : (Z>) ( -D) 1 CNT ! (S) BEGIN
 10   KEY DUP 13 = NOT WHILE ?Z IF
 11   DUP EMIT C, (S) 1 CNT +! ELSE
 12  20 = IF -1 ALLOT -1 CNT +! (S)
 13 -1 211 +! THEN THEN REPEAT DROP
 14   (Z> CNT @ 1 - MINUS DP +! ;
 15  -->

SCR # 114
  0 ( INPUT WORDS                 EF)
  1 : Z>P ( AN) (Z>) DROP SWAP
  2   PAD + ! ;
  3 : C> DUP EMIT ;
  4 : C>P ( A) KEY DUP 13 = IF DROP
  5   74 THEN  C> SWAP PAD + C! ;
  6 : PC! ( CA) PAD + C! ;
  7
  8
  9 -->
 10
 11
 12
 13
 14
 15

SCR # 115
  0 ( BUSI COMPUTER SPECIFIC      EF)
  1 HEX
  2 : NQ ' SPACE CFA ' QUIT A + ! ;
  3 : AQ ' CR CFA ' QUIT A + ! ;
  4 : CU CH CV ;
  5 : PRON 1 PR# ;
  6 : PROF 0 PR# ;
  7 : 3>CLR 3 22 C! HOME ;
  8 : CLR 0 22 C! HOME ;
  9 DECIMAL
 10 : ['] [COMPILE] ' ;
 11       -->
 12 ( FUER APPLE II COMPUTER
 13 3>CLR LOESCHT BILDSCHIRM
 14 AB DER DRITTEN ZEILE NACH UNTEN)
 15
```

```
SCR # 116
  0  ( BUSINESS ADDR INPUT cntd  ef)
  1
  2  0 VARIABLE #NR
  3  : (CL) ( n) 0 DO 32 EMIT LOOP ;
  4  : MC 16 2 CU ;
  5  : ?KEY ( -f) KEY DUP 13 = IF
  6    DROP 1  ELSE 74 = THEN ;
  7  : MORE? MC 19 (CL) MC
  8     ." MEHR ( /N)" ?KEY ;
  9  : OK? MC 35 (CL) MC
 10    ." OK ( /N) " ?KEY ;
 11  : 1S SPACE ; : 3S 3 SPACES ;
 12  : 5S 5 SPACES ;
 13
 14  -->
 15

SCR # 117
  0  (      VIRTUAL MEMORY         ef)
  1   80 CONSTANT START
  2   128 CONSTANT RECLEN
  3 : #INDEX ( n-a) RECLEN B/BUF
  4   */MOD START + DUP 356 = OVER
  5   356 > OR IF 2 +
  6   THEN BLOCK + ; ( ONLY C64)
  7  : FIRST# ( -a) 0 #INDEX ;
  8  : +NR 1 FIRST# +! UPDATE ;
  9  : !MEM PAD  FIRST# @ #INDEX
 10    RECLEN CMOVE UPDATE +NR ;
 11  : @MEM ( n) #INDEX PAD RECLEN
 12    CMOVE ;
 13 : !ENTRY PAD #NR @ #INDEX RECLEN
 14   CMOVE UPDATE FLUSH ;
 15   -->

SCR # 118
  0  ( BUSI DATUM INPUT           ef)
  1  0 VARIABLE DAT  6 ALLOT
  2  : DIN 13 WORD HERE COUNT ' DAT
  3  SWAP  CMOVE ;
  4  : D>S FIRST# 2 + ' DAT  SWAP 6
  5    CMOVE UPDATE ;
  6  : D<S FIRST# 2 + ' DAT  6
  7  CMOVE ;
  8  : WAS 16 2 CU ." WAS " 30 (CL)
  9  16 6 CU QUIT ;
 10
 11
 12
 13 : C<>C ' RECLEN ! ' START ! ;
 14   -->
 15
```

```
SCR # 119
  0  ( BUSI LAGER  INPUT cntd    ef)
  1   : ?N ( c-c/0) DUP 58 < IF DUP
  2  48 < IF DROP 0 THEN ELSE DROP
  3  0 THEN ;
  4 : +PAD ( n) HERE + 1 + ;
  5   : >N ( an) 0 0 +PAD CNT !
  6  DO DUP I + C@ ?N DUP IF CNT @
  7  C! 1 CNT +! ELSE DROP THEN LOOP
  8  32 CNT @ C! DROP ;
  9  (       VIRTUAL MEMORY         ef)
 10   : (IM) ( n) #INDEX RECLEN
 11  CMOVE UPDATE ;
 12   : (INR) ( n) FIRST# @ 2 * B/BUF
 13     /MOD 1 + BLOCK + ! UPDATE ;
 14   : @NR ( a-n) 2 * B/BUF /MOD 1 +
 15     BLOCK + @ ;    -->

SCR # 120
  0  ( BUSI LAGER EINGABE          ef)
  1 : N>Z ( an-d) DUP -1 +PAD !
  2     >N  -1 +PAD NUMBER  ;
  3 : INR PAD 4 N>Z DROP (INR) ;
  4 : #SUCH ( n-af) 1 SWAP FIRST#
  5  @ 1 DO DUP I @NR = IF DROP DROP
  6     I #NR ! I 0 LEAVE THEN LOOP ;
  7   : .- 45 EMIT ;
  8 : PAC ( na) PAD + SWAP 32 FILL ;
  9   : .-- ( n) 0 DO .- LOOP ;
 10 : .DATE ' DAT 6 N>Z #DA TYPE ;
 11 : (DATUM) DIN 1 9 CU
 12    8 SPACES 1 9 CU .DATE D>S ;
 13 : (?DATUM) D<S 1 9 CU .DATE ;
 14 -->
 15

SCR # 121
  0 ( INVENTORY CNTD          0131EF)
  1 : #! PAD 4 N>Z DROP #NR @ 2 *
  2    B/BUF /MOD 1+ BLOCK + !
  3    UPDATE ;
  4

SCR # 122
  0 ( COMMON SCREENS END          EF)
  1  : NET ( dn-d')  >R 1000
  2    UDN* R M/MOD ROT R> 2 / 1 -
  3    > IF 1.  D+ THEN ;
  4  : BRT ( dn-d') UDN* 1000 M/MOD
  5    ROT 499 > IF 1.  D+ THEN ;
  6  : DESCR 0 VARIABLE -2 ALLOT ;
  7
  8 124 LOAD
```

```
SCR # 124
  0  ( BUSI LAGER cntd            ef)
  1
  2  DESCR (NR) 0 , 4 ,
  3  DESCR (BZ) 4 , 20 ,
  4  DESCR (TX) 24 , 1 ,
  5  DESCR (MF) 25 , 2 ,
  6  DESCR (RL) 27 , 3 ,
  7  DESCR (VK) 30 , 8 ,
  8  DESCR (LM) 38 , 5 ,
  9  DESCR (VM) 43 , 5 ,
 10  DESCR (DA) 48 , 4 ,
 11  DESCR (EK) 52 , 8 ,
 12  DESCR (CI) 60 , 4 ,
 13
 14   -->
 15
```

```
SCR # 125
  0  ( BUSI LAGER cntd            ef)
  1  : NR (NR) 2@ ; : BZ (BZ) 2@ ;
  2  : TX (TX) 2@ ; : MF (MF) 2@ ;
  3  : RL (RL) 2@ ; : VK (VK) 2@ ;
  4  : LM (LM) 2@ ; : VM (VM) 2@ ;
  5  : DA (DA) 2@ ; : EK (EK) 2@ ;
  6  : CNR 4 18 CU ; : CBZ 5 18 CU ;
  7  : CTX 6 18 CU ; : CMF 7 18 CU ;
  8  : CRL 8 18 CU ; : CVK 9 18 CU ;
  9  : CVM 11 18 CU ;
 10  : CLM 10 18 CU ;
 11  : CDA 12 18 CU ;
 12  : CEK 13 18 CU ;
 13
 14    -->
 15
```

```
SCR # 126
  0  ( INVENTORY INPUT cntd       ef)
  1 VOCABULARY LAGER IMMEDIATE
  2 LAGER DEFINITIONS
  3
  4  : .CMD 18 0 CU 39 .-- 19 1 CU
  5  ." E)INGABE EN)DE ME)HR EX)IT S
  6 P)EICHERN"
  7  20 1 CU
  8  ." LA BE MW HE MM VKP LAM VKM L
  9 D EKP" ;
 10
 11
 12      -->
 13
 14
 15
```

```
SCR # 127
  0  ( INVENTORY INPUT           ef)
  1  : M1 ." LAGER#           : " ;
  2  : M2 ." BEZEICHNUNG      : " ;
  3  : M3 ." MWST CODE        : " ;
  4  : M4 ." HERSTELLER       : " ;
  5  : M5 ." MINDEST MENGE    : " ;
  6  : M6 ." VERKAUFSPREIS    : " ;
  7  : M7 ." LAGERMENGE       : " ;
  8  : M8 ." VERK. MENGE      : " ;
  9  : M9 ." LETZTER ZUGRIFF: " ;
 10  : M10 ." EINKAUFSPREIS   : " ;
 11  : .ST ( n) 0 DO 46 EMIT
 12    LOOP ;
 13
 14       -->
 15

SCR # 128
  0  ( INVENTORY INPUT cntd      ef)
  1
  2
  3  : MASK 3>CLR NQ
  4    4 1 CU M1 NR DROP .ST
  5    5 1 CU M2 BZ DROP .ST
  6    6 1 CU M3 TX DROP .ST
  7    7 1 CU M4 MF DROP .ST
  8    8 1 CU M5 RL DROP .ST
  9    9 1 CU M6 VK DROP .ST
 10   10 1 CU M7 LM DROP .ST
 11   11 1 CU M8 VM DROP .ST
 12   12 1 CU M9 DA DROP .ST
 13   13 1 CU M10 EK DROP .ST
 14   .CMD ;  -->
 15

SCR # 129
  0 ( INVENTORY INPUT cntd      ef)
  1 : CLP ( na) PAD + SWAP 32 FILL ;
  2  : LA NR CLP CNR NR IP WAS ;
  3  : BE BZ CLP CBZ BZ IP WAS ;
  4  : MW TX CLP CTX TX IP WAS ;
  5  : HE MF CLP CMF MF IP WAS ;
  6  : MM RL CLP CRL RL IP WAS ;
  7  : VKP VK CLP CVK VK IP WAS ;
  8  : LAM LM CLP CLM LM IP WAS ;
  9  : VKM VM CLP CVM VM IP WAS ;
 10  : LD DA CLP CDA DA IP WAS ;
 11  : EKP EK CLP CEK EK IP WAS ;
 12  : E CNR NR IP CBZ BZ IP CTX
 13  TX IP CMF MF IP CRL RL IP CVK
 14  VK IP CLM LM IP CEK EK IP WAS ;
 15  -->
```

```
SCR # 130
  0  ( INVENTORY INPUT cntd       ef)
  1  : RMASK
  2    4 18 CU  NR DROP .ST
  3    5 18 CU  BZ DROP .ST
  4    6 18 CU  TX DROP .ST
  5    7 18 CU  MF DROP .ST
  6    8 18 CU  RL DROP .ST
  7    9 18 CU  VK DROP .ST
  8   10 18 CU  LM DROP .ST
  9   11 18 CU  VM DROP .ST
 10   12 18 CU  DA DROP .ST
 11   13 18 CU  EK DROP .ST ;
 12
 13
 14     -->
 15

SCR # 131
  0  ( BUSI LAGER MENUE           ef)
  1  : .MSG 3 10 CU ." LAGERVERWALTU
  2 NG"  ;
  3  : .MSG1 3>CLR  .MSG
  4   5 9 CU ." RUN"
  5   6 9 CU ." DATUM TTMMYY"
  6   7 9 CU ." EINGABE"
  7   8 9 CU ." ABGANG"
  8   9 9 CU ." ZUGANG"
  9  10 9 CU ." DRUCKER"
 10  11 9 CU ." <#> AUSGABE"
 11  12 9 CU ." NEU"
 12  13 9 CU ." FIN"
 13  14 9 CU ." INHALT"
 14  15 9 CU ." BACKUP"
 15  WAS ;  -->

SCR # 132
  0  ( BUSI LAGER EINGABE         ef)
  1  : NEU ." WIRKLICH? (J/N)" KEY
  2  74 = IF  1 FIRST# ! UPDATE THEN
  3    WAS ;
  4
  5  : EINGABE PADC MASK WAS ;
  6  : EN INR IMEM FLUSH .MSG1  ;
  7  : EX FLUSH .MSG1 ;
  8  : ME INR IMEM PADC RMASK WAS ;
  9  : SP #I IENTRY .MSG1 ;
 10
 11
 12
 13  -->
 14
 15
```

```
SCR # 133
  0  ( BUSI LAGER SUCHE           ef)
  1  : (NIL) ." NICHT IN LISTE" ;
  2  : EOL ." ENDE DER LISTE" ;
  3  : NIL ( n) DROP 15 1 CU (NIL)
  4    KEY DROP ;
  5  : DATUM (DATUM) .MSG1 ;
  6  : ?DATUM (?DATUM) .MSG1 ;
  7  : FIN FLUSH .MSG1 QUIT ;
  8  : RUN FLUSH  CLR 1 2 CU ." DATU
  9 M:" NQ 80 64 C<>C ?DATUM ;
 10
 11 : BACKUP EDITOR 40 64 DUPLICATE
 12    65 89 DUPLICATE 90 114
 13    DUPLICATE 115 139 DUPLICATE
 14    140 160 DUPLICATE RUN ;
 15  FORTH -->

SCR # 134
  0 ( BUSI LAGER AUSGABE          ef)
  1  : .L ( na) PAD + SWAP -TRAILING
  2    ATYPE ;
  3  : (DRUCK) MASK      CNR NR .L
  4  CBZ BZ .L CTX TX .L CMF MF .L
  5  CRL RL .L CVK VK .L CLM LM .L
  6  CVM VM .L CDA DA .L CEK EK .L ;
  7
  8  : AUSGABE ( n) 3>CLR #SUCH IF
  9    NIL ELSE @MEM (DRUCK) THEN
 10    WAS ;
 11
 12
 13
 14  -->
 15

SCR # 135
  0  ( BUSI ZUGANG                ef)
  1  : MC1 16 2 CU ;
  2  : LMC 10 18 CU LM DROP .ST ;
  3  : VMC 11 18 CU VM DROP .ST ;
  4  : DAC 12 18 CU DA DROP .ST ;
  5  : CMB MC1 35 (CL) ;
  6  : (ZUGANG)        MC1
  7  ." LAGER #:    " PAD 5 EXPECT
  8  PAD 5 N>Z DROP ;
  9  : (ELM) CMB MC1 ." MENGE "
 10  ." ZUGANG:" LM IPP ;
 11  : LM>Z LM PAD + SWAP N>Z ;
 12  : LME>Z LM PADD + SWAP N>Z ;
 13  : LM> LM PAC  #N LM >R DROP R>
 14    PAD + SWAP  CMOVE ; -->
 15
```

```
SCR # 136
  0  ( BUSI ZUGANG cntd             ef)
  1  : .LM LMC  CLM LM .L ;
  2  : LM+ LM>Z LME>Z D+ LM> .LM ;
  3  : VME>Z VM PADD + SWAP N>Z ;
  4  : LM- LM>Z VME>Z DMINUS D+
  5  2DUP 0 < IF DROP 2DROP 0 0 ELSE
  6  DROP THEN LM> .LM ;
  7  : VM>Z VM PAD + SWAP N>Z ;
  8  : VM> VM PAC  #N VM PAD +
  9  >R DROP R> SWAP CMOVE ;
 10  : DA> DA PAD + SWAP FIRST# 2 +
 11  ROT  ROT CMOVE ;
 12  : .VM VMC  CVM VM .L ;
 13  : .DA DAC CDA DA .L ;
 14  : VM+ VM>Z VME>Z D+ VM> .VM
 15    LM- .LM DA> .DA ; -->

SCR # 137
  0  ( BUSI ZUGANG cntd             ef)
  1  : ?INL ( n-n'f) (ZUGANG) #SUCH
  2    IF CMB NIL WAS   ELSE
  3    @MEM (DRUCK) THEN ;
  4  : (ZU) ?INL (ELM) LM+         ;
  5  : ISOK PAD #NR @ (IM) WAS ;
  6
  7  : ZUGANG (ZU) ISOK ;
  8  : (EVM) CMB MC1 ." MENGE "
  9    ." ABGANG:" VM IPP ;
 10
 11  : (AB) ?INL (EVM) VM+ ;
 12  : ABGANG (AB) ISOK ;
 13  : (N>Z) PAD + SWAP N>Z DROP ;
 14  : NR>Z NR (N>Z) ;
 15  : RL>Z RL (N>Z) ;  -->

SCR # 138
  0  ( BUSI DRUCKER AUSGABE        ef)
  1
  2  : RL?  LM>Z RL>Z 0 DMINUS D+
  3    0< IF 42 EMIT   THEN DROP ;
  4  : (RP) ( na-n') 0 ROT ROT PAD
  5    + DUP   >R + R> DO I C@ DUP
  6  0= IF DROP  LEAVE ELSE 32 = IF
  7  LEAVE ELSE 1 + THEN THEN LOOP ;
  8
  9  -->
 10
 11
 12
 13
 14
 15
```

```
SCR # 139
  0  ( BUSI DRUCKAUSGABE cntd    ef)
  1  : (AUS) 0 H ! NR>Z 0 #N 0  5
  2   (RADJ) BZ  7 PTEX> TX 28 PTEX>
  3   MF 30 PTEX> RL>Z 0 #N 33 4
  4   (RADJ) VK (N>Z) 0 #DM 40 7
  5   (RADJ) LM>Z #N 49 5 (RADJ)
  6   VM>Z #N 55 5 (RADJ) DA 61
  7   PTEX> EK (N>Z) 0 #DM 67 7
  8   (RADJ) 1S RL? ;
  9
 10 : (SHW) 4 H ! PAD 4 N>Z DROP
 11   4 .R BZ 5 PTEX> LM>Z #N 26 5
 12   (RADJ) VK (N>Z) 0 #DM 32 7
 13   (RADJ) ;
 14
 15  -->

SCR # 140
  0 ( BUSI INHALT               131 EF)
  1 : INHALT 0 V ! 3>CLR 3 0 CU
  2   FIRST# @ 1 DO I @MEM (SHW) CR
  3   1 V +! V @ 16 = IF KEY DROP
  4   3>CLR 3 0 CU 0 V ! THEN LOOP
  5   24 2 CU ." BELEIBIGE TASTE DRU
  6 ECKEN" KEY DROP .MSG1 ;
  7
  8
  9


SCR # 141
  0  ( BUSI DRUCKAUSGABE cntd    ef)
  1  : .SP 0 DO 32 EMIT   LOOP ;
  2  : 3CR 3 0 DO CR  LOOP ;
  3  : .D ' DAT 6 N>Z #DA ATYPE ;
  4
  5  : (HEAD) CR 12 .SP ." LAGERLIST
  6 E "      .D 3CR 4 .SP ." # " 1S
  7  ." DE" 19 .SP ." C" 1S ." MF"
  8  3 .SP ." RL" 6 .SP ." SP" 6 .SP
  9  ." QU" 5 .SP ." SD"  1S
 10  ." DA" 7 .SP ." PR" CR ;
 11  : HEAD PADC (HEAD) ;
 12  : AUS CR HEAD     CR   FIRST#
 13   @ 1 DO I @MEM (AUS) CR  LOOP ;
 14  : DRUCKER  PRON AUS PROF RUN ;
 15 150 LOAD

SCR # 150
  0 FORTH DEFINITIONS -->
  1
```

```
SCR # 151
  0 ( BUSINESS ADDR INPUT        ef)
  1 VOCABULARY ADRESSEN IMMEDIATE
  2 ADRESSEN DEFINITIONS
  3   DESCR   (VN) 0 , 28 ,
  4   DESCR   (CO) 28 , 23 ,
  5   DESCR   (ST) 51 , 23 ,
  6   DESCR  (PLZ) 74 ,  7 ,
  7   DESCR  (ORT) 81 , 21 ,
  8   DESCR   (NR) 102 , 5 ,
  9   DESCR   (AM) 107 , 7 ,
 10   DESCR  (TEL) 114 , 11 ,
 11   DESCR   (IC) 124 , 2 ,
 12  : .MSG3 2 2 CU
 13  ." AENDERE <BZ>  LOESCHEN"
 14  ."  EINTRAG RUN" ;
 15     -->

SCR # 152
  0  ( BUSINESS ADDR INPUT cntd  ef)
  1  : VN (VN) 2@ ;
  2  : CO (CO) 2@ ; : ST (ST) 2@ ;
  3  : ORT (ORT) 2@ ; : C1 (NR) 2@ ;
  4  : PLZ (PLZ) 2@ ; : C2 (AM) 2@ ;
  5  : TEL (TEL) 2@ ; : IC (IC) 2@ ;
  6
  7  : CVN 6 4 CU ;
  8  : CCO 8 4 CU ; : CST 10 4 CU ;
  9  : CPL 12 4 CU ;
 10  : COR 12 12 CU ;
 11  : CC1 14 4 CU ;
 12  : CC2 14 10 CU ;
 13 : CTE 14 18 CU ;
 14
 15                             -->

SCR # 153
  0  ( BUSINESS ADDR INPUT cntd  ef)
  1  : (INPUT) #NR @ 3 4 CU .
  2  CVN VN IP
  3  CCO CO IP CST ST IP CPL PLZ IP
  4  COR ORT IP CC1 C1 IP CC2 C2 IP
  5  CTE TEL IP ;
  6   : FNC CVN 28 (CL) ;
  7   : COC CCO 23 (CL) ;
  8   : STC CST 23 (CL) ;
  9   : CYC COR 21 (CL) ;
 10   : CTC CPL  7 (CL) ;
 11   : NRC CC1  5 (CL) ;
 12   : AMC CC2  7 (CL) ;
 13   : TEC CTE 11 (CL) ;
 14   : CL FNC     COC STC CTC
 15     CYC NRC AMC TEC ; -->
```

```
SCR # 154
  0 -->
  1

SCR # 155
  0 -->
  1

SCR # 156
  0  ( BUSINESS MASK  cntd        ef)
  1  : .I 0 DO I 1+ 10 MOD   48 +
  2     EMIT   LOOP ;
  3 : 1R ( n) 3 CU 38 3 DO .- LOOP ;
  4  : 2R 6 3 CU .- 6 32 CU 6 .— ;
  5 : 3R ( n) DUP 3 CU .- 27 CU
  6   11 .— ;
  7  : 4R 12 3 CU .- 12 11 CU .-
  8   12 33 CU 5 .— ;
  9 : 5R 14 3 CU .- 14 9 CU .- 14 17
 10   CU .- 14 29 CU 6 .— 14 37 CU
 11    .- ;
 12  : 6R 16 4 CU 5 .I .- 7 .I
 13     .- 11 .I  .- ;
 14
 15                         -->

SCR # 157
  0  ( BUSINESS ADDR INPUT cntd  ef)
  1  : MASK CLR 5 1R 2R 7 1R 8 3R 9
  2    1R 10 3R 11 1R 4R 13 1R 5R
  3    15 1R 6R ;
  4  : .ADDR CR 3S VN PRINT CR
  5   3S CO PRINT CR 3S ST PRINT CR
  6   3S PLZ PRINT 1S ORT PRINT CR
  7   3S C1 PRINT 1S C2 PRINT CR
  8   3S TEL PRINT ;
  9  : .MAD CLR MASK CVN VN PRINT
 10  CCO CO PRINT CST ST PRINT
 11  CPL PLZ PRINT COR ORT PRINT
 12  CC1 C1 PRINT CC2 C2 PRINT
 13  CTE TEL PRINT ;
 14  : .MSG 2 10 CU   ." ADRESS-VERW
 15 ALTUNG" ;   -->

SCR # 158
  0  ( BUSINESS ADDR INPUT cntd  ef)
  1  : .MSG1 CLR .MSG 4 2 CU
  2  ." TERMINAL EINGABE"  5 11 CU
  3  ." SUCHE   <BZ> <NAME>" 6  5 CU
  4  ." <NR>  FINDE"  7 11 CU
  5  ." INHALT"
```

```
  6   10 2 CU
  7  ." DRUCKER  LABEL" 11 11 CU
  8  ." SUCHE   <BZ> <NAME>"
  9  12 11 CU ." INHALT "
 10  13 5 CU ." <NR>  DRUCKE"
 11
 12  15 2  CU ." RUN" 16 2 CU
 13  ." NEU " 17 2 CU ." RECHNUNG"
 14   18 2 CU ." WAS " QUIT ; —>
 15

SCR # 159
  0  ( BUSINESS SEARCH            ef)
  1  0 VARIABLE WO
  2 : ['] [COMPILE] ' ;
  3  : (VERGL) ( aa'c-f)
  4    BEGIN ROT DUP C@ >R OVER I =
  5          R> SWAP DUP IF 0
  6          ELSE DROP >R ROT DUP C@
  7          R> = DUP DUP THEN WHILE
  8          2DROP 1+ >R 1+ R> ROT
  9    REPEAT >R 2DROP 2DROP R> ;
 10  : WHAT ['] 2 - EXECUTE SWAP
 11   DROP DUP WO ! 13 WORD HERE
 12   COUNT ROT PAD + SWAP CMOVE ;
 13   —>
 14
 15

SCR # 160
  0  ( BUSINESS SEARCH  cntd     ef)
  1  : VERGL PAD WO @ + #NR @ #INDEX
  2          WO @ + 32 (VERGL) ;
  3  : NIL CR ." NICHT IN LISTE " ;
  4  : EOL CR ." ENDE DER LISTE " ;
  5  : .NAME #NR @ @MEM .ADDR ;
  6
  7  : INPUT MASK PADC .MSG BEGIN
  8    FIRST# @     #NR !
  9    (INPUT) OK? IF !MEM
 10    THEN  MORE?
 11    WHILE CL PADC MC 19 (CL)
 12    REPEAT FLUSH .MSG1 ;
 13    —>
 14
 15

SCR # 161
  0  ( BUSINESS OUTPUT           ef)
  1  : 3? ( -f) CNT @ 3 = ;
  2  : (CONTENT) ( n)
  3    DUP . @MEM .ADDR
```

```
  4    1 CNT +! 3? IF KEY 0 CNT !
  5    CLR 32 = 1 XOR IF .MSG1
  6    THEN THEN ;
  7  : .CONTENT CLR    0 CNT !
  8     CR FIRST# @ DUP 1 = 1 XOR
  9    IF 1 DO I (CONTENT)
 10    CR LOOP THEN EOL KEY DROP
 11    .MSG1 ;
 12
 13
 14
 15  -->

SCR # 162
  0  ( BUSINESS SEARCHING cntd   ef)
  1  : MOVE> PAD PAD 128 + RECLEN
  2    CMOVE ;
  3  : <MOVE PAD 128 + PAD RECLEN
  4    CMOVE ;
  5  : FOUND MOVE> #NR @ (CONTENT)
  6     CR   <MOVE ;
  7  : (SEARCH) DO I #NR ! VERGL
  8   IF FOUND DROP 1 THEN LOOP ;
  9
 10  : SEARCH 0 CNT ! PADC CLR
 11    WHAT 0 FIRST# @ 1 (SEARCH)
 12    IF  EOL ELSE NIL THEN KEY
 13    DROP .MSG1 ;
 14  -->
 15

SCR # 163
  0  ( BUSINESS DELETING         ef)
  1
  2
  3  -->
  4
SCR # 164
  0  ( BUSINESS ENTRY             ef)
  1 ADRESSEN DEFINITIONS
  2
  3  : (ENTRY) CLR MASK PADC  BEGIN
  4    (INPUT) OK? 1 XOR WHILE
  5    CL REPEAT !ENTRY ;
  6  : ENTRY  (ENTRY) .MSG1 ;
  7  : (FINDE) ( n) DUP #NR ! DUP
  8    @MEM .MAD 4 4 CU .  ;
  9  : FINDE  (FINDE) .MSG3 WAS ;
 10
 11  : .* PAD RECLEN 32 FILL ;
 12
 13  -->
 14
```

```
SCR # 165
  0  ( BUSINESS TRANSLATION      ef)
  1  : EINGABE INPUT ;
  2  : EINTRAG ENTRY ;
  3  : INHALT .CONTENT ;
  4
  5
  6
  7
  8  : SUCHE SEARCH ;
  9  : LOESCHEN  .* !ENTRY WAS ;
 10
 11
 12
 13
 14  -->
 15
SCR # 166
  0  ( BUS. MAILING LABEL PRINT  ef)
  1  36 CONSTANT PH
  2   8 CONSTANT PV
  3
  4  : TABS ( nn') SWAP - 0 DO SPACE
  5    LOOP ;
  6  : HTAB ( n)  PH TABS ;
  7  : (DR) ( naa'-n') + SWAP
  8  -TRAILING DUP ROT SWAP ATYPE ;
  9  : SPAD ( a)  DUP  RECLEN
 10    1 + + SWAP DO I C@ 0=
 11    IF 32 I  C! THEN LOOP ;
 12  : SS 2 SPACES ;
 13  : NCR ( n) 0 DO CR LOOP ;
 14  : 0Z 0 #N DUP >R ATYPE R> HTAB
 15    0 #N ATYPE ;    -->

SCR # 167
  0  ( BUSI. PRINTING cntd       ef)
  1  : 1Z PAD SPAD VN PAD (DR)
  2    HTAB VN PADD (DR) DROP ;
  3  : 2Z PAD SPAD CO PAD (DR) HTAB
  4    CO PADD (DR) DROP ;
  5  : 3Z PAD SPAD ST PAD (DR) HTAB
  6    ST PADD (DR) DROP ;
  7  : 4Z PAD SPAD PLZ PAD (DR) 10
  8   TABS 10 ORT PAD (DR) + HTAB
  9   PLZ PADD (DR) 10 TABS ORT
 10   PADD (DR) DROP ;
 11  : 5Z PAD SPAD C1 PAD (DR) 1S
 12 1 + C2 PAD (DR) + 1S 1 + TEL PAD
 13  (DR) + HTAB C1 PADD (DR) 1S
 14  C2 PADD (DR) 1S TEL PADD (DR)
 15  2DROP DROP ;  -->
```

```
SCR # 168
  0  ( BUSI. PRINTING cntd        ef)
  1  : DRUCKE SS 0Z CR SS 1Z CR  SS
  2     2Z  CR SS 3Z CR CR 4Z CR  ;
  3  : (MEM@) CNT @ DUP @MEM
  4     1 CNT +! ;
  5  : MEM@ ( a) BEGIN (MEM@) PAD C@
  6    42 = DUP IF SWAP  DROP THEN
  7    NOT UNTIL ;
  8  : FIN? ( -f) CNT @ FIRST# @ < ;
  9  : (LABEL)          BEGIN FIN?
 10  WHILE MEM@ MOVE> FIN? NOT IF 0
 11  PADC DRUCKE 3 NCR ELSE MEM@
 12  DRUCKE 3 NCR THEN REPEAT ;
 13  : .LABEL PRON 1 CNT !  (LABEL)
 14    PROF  .MSG1 ;  —>
 15

SCR # 169
  0  ( BUSI PRINT SEARCH          ef)
  1  : @MN #NR @ DUP @MEM 1 CNT +! ;
  2  : RC RECLEN ;
  3  : M> PAD PAD 256 + RC CMOVE ;
  4  : <M PAD 256 + PAD RC CMOVE ;
  5  : CNT0 ( -f) CNT @ 2 MOD 0= ;
  6  : .SUCHE  M> @MN
  7     CNT0 IF MOVE>  ELSE DRUCKE
  8     3 NCR THEN <M ;
  9  : ((SUCH)) DO I #NR ! VERGL
 10     IF .SUCHE THEN LOOP ;
 11
 12  : (SUCHE) 1 CNT ! PADC PRON
 13     WHAT 0 FIRST# @ 1 ((SUCH))
 14     CNT0 IF 0 DRUCKE THEN
 15     PROF .MSG1 QUIT   ;  —>

SCR # 170
  0  ( BUSI .INHALT               ef)
  1
  2  : LL 64 0 DO .- LOOP ;
  3
  4  : IDRUCK DRUCKE 5Z CR LL CR ;
  5  : (.INHALT) 1 CNT ! BEGIN FIN?
  6  WHILE MEM@ MOVE> FIN? NOT IF 0
  7  PADC IDRUCK ELSE MEM@ IDRUCK
  8  THEN REPEAT ;
  9  : .INHALT PRON  (.INHALT)
 10   PROF .MSG1 ;
 11
 12 : C>L 80 64 C<>C ;
 13 : C>A 320 128 C<>C ;
 14
 15  —>
```

```
SCR # 171
  0  ( BUSI VOCABULARY DRUCKER    ef)
  1  VOCABULARY DRUCKER IMMEDIATE
  2  DRUCKER DEFINITIONS
  3  : LABEL .LABEL ;
  4  : INHALT .INHALT ;
  5  : SUCHE (SUCHE) ;
  6  ADRESSEN DEFINITIONS
  7  VOCABULARY TERMINAL IMMEDIATE
  8  TERMINAL DEFINITIONS
  9  : INHALT .CONTENT ;
 10  : SUCHE SEARCH ;
 11  ADRESSEN DEFINITIONS
 12  : RUN CLR NQ    EMPTY-BUFFERS
 13  C>A .MSG1 ;
 14  : NEU 1 FIRST# ! UPDATE FLUSH
 15  .MSG1 ;  -->

SCR # 172
  0  ( BUSI AENDERN                 ef)
  1  : (AEND) ['] 2 - EXECUTE CR
  2    2DUP PAC
  3    OVER   3 + 19 SWAP CU
  4    60 EMIT 19 3 CU ?IP  ;
  5
  6  : AENDERE (AEND) .MAD OK?
  7    IF !ENTRY THEN WAS ;
  8
  9  : DRUCKE ( n) CNT ! PRON
 10    (LABEL) PROF .MSG1 ;
 11
 12 0 VARIABLE BLNR
 13
 14
 15 180 LOAD

SCR # 180
  0 ( INVOICING BEGINS              EF)
  1 ADRESSEN DEFINITIONS
  2 37 BLNR ! 1 VARIABLE BEST
  3 30 VARIABLE CNT#
  4 : (J/N) ( -f) ." ( /N)" KEY DUP
  5 74 = OVER 13 = OR OVER 88 = IF
  6 2DROP WAS ELSE SWAP DROP THEN ;
  7  : .CM1 18 0 CU 39 .-
  8  19 3 CU ." ANSCHRIFT"
  9  19 18 CU ." BES)TELLUNG"
 10  21 3 CU ." SON)DERARTIKEL"
 11  21 18 CU ." WEITER"
 12  23 3 CU ." ENDE" ;
 13
 14  : IN ( n)  FIRST# @ SWAP - PAD
 15    ! ;  -->
```

```
SCR # 181
  0  ( BUSI INVOICING cntd        ef)
  1  : JCL ( n) DUP 16 CU 9 (CL) 16
  2    CU ;
  3  : (ADR) MASK PADC FIRST# @
  4    #NR ! (INPUT) ;
  5  : SHOW CLR 3 2 CU MOVE>
  6    @MEM .ADDR <MOVE ;
  7  : ADR BEGIN CL (ADR) OK? IF
  8     !MEM 1 ELSE 0 THEN UNTIL ;
  9  : KDNR BEGIN   3 2 CU
 10  ." KUNDEN-NR   :" (Z>) DROP
 11  DUP IF DUP PAD ! SHOW OK? IF 1
 12  ELSE 0 THEN ELSE DROP ADR PADC
 13  FLUSH 1 IN 1 THEN UNTIL CLR ;
 14    —>
 15

SCR # 182
  0  ( BUSI INVOICING cntd        ef)
  1
  2 : .ST 0 DO 46 EMIT LOOP ;
  3  : ONR 4 2 CU
  4  ." BESTELL-NR    :" 4 16 CU
  5  10 .ST 4 16 CU 10 2 IP ;
  6  : ACC 5 2 CU
  7  ." BESTELL DATUM:" 5 16 CU
  8   6 .ST 5 16 CU 6 12 IP ;
  9
 10   —>
 11
 12
 13
 14
 15

SCR # 183
  0  ( BUSI INVOICING cntd        ef)
  1  : SOND 8 2 CU
  2  ." SONDER-RABATT:"
  3  (J/N) 8 JCL IF 89 19 PC! 20 Z>P
  4   ELSE 65 C> 19 PC! THEN ;
  5
  6  : RA 7 2 CU
  7  ." RABATT ( /N) :"
  8  (J/N) 7 JCL IF 74 EMIT SOND
  9  ELSE 78 C> 19 PC!  THEN ;
 10
 11  : TAXE 6 2 CU
 12  ." STEUER ( /N) :"  22 C>P ;
 13  : PAD! PAD BLNR @ BLOCK B/BUF
 14    CMOVE  UPDATE ;
 15  —>
```

```
SCR # 184
  0  ( BUSI INVOICING cntd        ef)
  1
  2
  3
  4
  5
  6  : VSK 11 2 CU
  7  ." VERSANDKOSTEN:" 25 Z>P ;
  8
  9  : ANSCHRIFT  CLR PADC .CM1
 10   30 CNT# ! KDNR
 11   BEGIN ONR ACC TAXE
 12  RA   VSK MC OK? UNTIL
 13  PAD! CLR .CM1  WAS ;     -->
 14
 15
```

```
SCR # 185
  0  ( BUSI INVOICING cntd        ef)
  1
  2 : .MM 5 2 CU ." BEST. MENGE : "
  3        6 2 CU ." GEL.  MENGE : "
  4     7 2 CU ." BESTELL NR   : " ;
  5  : >OF ( n) CNT# @   Z>P
  6    CNT# +! ;
  7  : (ORD) 5 15 CU 2 >OF 6 15 CU
  8    2 >OF 7 15 CU 2 >OF ;
  9
 10  : (JC) 15 CU 10 (CL) ;
 11  : OFCL 5 (JC) 6 (JC) 7 (JC) ;
 12  : EOB ( n) B/BUF CNT# @ - > IF
 13  0 ." KEIN PLATZ MEHR" ELSE 1
 14  THEN ;
 15         -->
```

```
SCR # 186
  0  ( BUSI INVOICING cntd        ef)
  1
  2
  3  : BES CLR .CM1 BEGIN
  4   BEGIN OFCL .MM (ORD) OK?
  5   IF 1 ELSE -6 CNT# +! 0 THEN
  6   UNTIL MORE? OFCL 2 EOB AND NOT
  7   UNTIL -100 PAD CNT# @ + ! PAD!
  8   .CM1 WAS ;
  9 : WEITER FLUSH CLR .CM1 1 BLNR
 10   +! 1 BEST +! WAS ;
 11
 12
 13  -->
 14
 15
```

```
SCR # 187
  0  ( BUSI INVOICING cntd        ef)
  1
  2  : HMSK  ( n) DUP 2 CU 2 .ST
  3  DUP 5 CU 20 .ST DUP 26 CU 1 .ST
  4  28 CU 7 .ST ;
  5  : >BL ( n) DUP CNT# @ DUP ROT +
  6  CNT# ! IP ;
  7  :  BZ)        20 >BL ;
  8  :  TX)         1 >BL ;
  9  :  PR)         7 >BL ;
 10  : HEIN ( n) DUP 2 CU (Z>) DROP
 11    MINUS CNT# @ PAD + !
 12    2 CNT# +! DUP 5 CU  BZ)
 13  DUP 26 CU  TX)  28 CU  PR) ;
 14   —>
 15

SCR # 188
  0  ( BUSI INVOICING cntd        ef)
  1 ADRESSEN DEFINITIONS
  2 : SON CLR 30 EOB IF 3 BEGIN 1+
  3   DUP HMSK DUP HEIN OK? IF
  4   MORE? 30 EOB AND NOT ELSE
  5   -30 CNT# +! 0
  6   THEN UNTIL DROP -100 PAD CNT#
  7   @ + ! PAD! 3>CLR .CM1 WAS
  8   THEN ;
  9 1070 CONSTANT MW1
 10 1140 CONSTANT MW2
 11 : CMW! ( N) DUP ' MW2 ! 1000 -
 12   2 / 1000 + ' MW1 ! ;
 13 : #EIN ( -D) 13 WORD HERE DUP C@
 14   + 1+ 32 SWAP C! HERE NUMBER ;
 15 —>

SCR # 189
  0  ( INVOICING START SCREEN    ef)
  1  : ?DATE C>L FIRST# 2 + 5 2 CU
  2  ." DATUM :" 6 N>Z #DA TYPE ;
  3  : DATUM C>L DIN D>S ?DATE WAS ;
  4  : (TX C>L FIRST# 10 + ;
  5  : @TX (TX @ ;
  6  : (.TX) @TX 1000 - 0 #% TYPE ;
  7 : .TX 6 2 CU ." MWST  :" (.TX) ;
  8  : MWST #EIN DROP 1000 + (TX !
  9  UPDATE @TX CMW! .TX TYPE WAS ;
 10  : I# C>L  FIRST# 8 + ;
 11  : @I# I# @ ; : (.I#) @I# 0 #N ;
 12  : .I# 7 2 CU ." RECHN#:"
 13   (.I#) ;
 14  : RECHN# #EIN DROP  I# !
 15    UPDATE .I# TYPE WAS ; —>
```

```
SCR # 190
  0  ( INVOICING START cntd      ef)
  1  : [.INV] 9 2 CU
  2  ." BSTF NEUER BESTELL-FILE"
  3  10 2 CU
  4  ." CONT BESTELL-FILE ERGAENZEN"
  5  11 2 CU
  6  ." RESS RECHNUNGEN SCHREIBEN"
  7
  8  16 2 CU ." WAS  " ;
  9
 10  : .INV CLR 1 4 CU
 11   ." RECHNUNGEN SCHREIBEN"
 12   ?DATE .TX .I# TYPE [.INV]
 13  QUIT ;
 14 : ENDE FLUSH .INV ;
 15  -->

SCR # 191
  0  ( BUSI     PRINTING         ef)
  1 ADRESSEN DEFINITIONS
  2 : DPRINT ( NABN') >R  BLOCK +
  3   R> + SWAP -TRAILING ATYPE ;
  4 : DTEX> ( NABN'N") ADJ DPRINT ;
  5 : 0ST START 0 ;
  6 : 1ST START RECLEN ;
  7 : 1Z VN 1ST 6 DTEX> CO 1ST
  8   40 DTEX> ;
  9 : 2Z ST 1ST 6 DTEX> PLZ 1ST
 10   40 DTEX> ORT 1ST 48 DTEX> ;
 11 : 3Z 40 ADJ 7 0 318 0 DPRINT
 12   TEL 1ST 50 DTEX> ;
 13 : 4Z 40 ADJ 6 0 318 16 DPRINT
 14   C2 1ST 50 DTEX> ;
 15 -->

SCR # 192
  0 ( RECHN. SCHREIBEN DRUCKAUSGABE)
  1 : .HE 1 VE 1Z ?TERMINAL IF
  2   KEY DROP THEN
  3   2 VE 2Z 1 VE 3Z 1 VE 4Z ;
  4 : @OF BLNR @ BLOCK PADD RECLEN
  5    CMOVE ;
  6
  7 : @AD PADD @ @MEM ;
  8
  9 -->
 10
 11
 12
 13
 14
 15
```

```
SCR # 193
  0 ( RECHN. SCHREIBEN CNTD      EF)
  1 : 5Z VN 6 PTEX> ;
  2 : 6Z CO 6 PTEX> C>L (.I#) 45
  3  TEX> ' DAT 6 N>Z #DA 64 TEX> ;
  4 : 7Z ST 6 PTEX> ;
  5 : 8Z PLZ 6 PTEX> ORT 15 PTEX> ;
  6 : 9Z PADD 12 + 6 N>Z #DA 45 TEX>
  7   PADD 2+ 10 57 TEX> ;
  8
  9 : .AD @OF C>A @AD 2 VE 5Z 1 VE
 10   6Z 1 VE 7Z 1 VE 8Z 1 VE 9Z ;
 11
 12
 13
 14
 15 -->

SCR # 194
  0  ( BUSI ARTIKEL DRUCKEN      ef)
  1  : SA ( a-a') PADD CNT# @ + + ;
  2  : @MM ( n-n') SA @ ;
  3  : .OM 0 @MM 0 #N 11 6 (RADJ) ;
  4  : .LM 2 @MM 0 #N 18 4 (RADJ) ;
  5  : .BN 4 @MM 0 #N 6 5 (RADJ) ;
  6  : .BZ BZ 24 PTEX> ;
  7  : .VP VK PAD + SWAP  N>Z #DM
  8    49 8 (RADJ) ;
  9 0 VARIABLE ANZ 0 VARIABLE RAB
 10 0 VARIABLE ZS  2 ALLOT
 11 0 VARIABLE 1MW 2 ALLOT
 12 0 VARIABLE 2MW 2 ALLOT
 13
 14
 15  -->

SCR # 195
  0  ( BUSI ARTIKEL DRUCKEN      ef)
  1 ADRESSEN DEFINITIONS
  2 : .NE ( D) #DM 58 7 (RADJ) ;
  3 : .SU ( D) #DM 65 8 (RADJ) ;
  4 : P>Z PAD 30 + 8 N>Z ;
  5 : MES ( NN') C>A SWAP 32 * START
  6   1 - BLOCK DUP C@ >R 1+ + R>
  7   ROT TEX> ;
  8  : MES1 1 23 MES 1 VE 2 23 MES ;
  9  : MES2 3 23 MES ;
 10 : ?LM ( -f)  2 @MM 0= NOT ;
 11 : .PD ?LM  IF 1 .BZ .VP ELSE 0
 12   MES1 THEN ;
 13 : ?TX ( -F) PADD 22 + C@ 74 = ;
 14  -->
 15
```

```
SCR # 196
  0  ( BUSI SUMMIERUNG            ef)
  1  : #LL ( -d) LM PAD + SWAP N>Z ;
  2  : #VV ( -d) VM PAD + SWAP N>Z ;
  3  : NLL ( d-d') #LL 2 @MM 0
  4  DMINUS D+ 2DUP 0< IF DROP 2DROP
  5  0. ELSE DROP THEN ;
  6  : NVV ( d-d') #VV 2 @MM 0 D+ ;
  7  : >LL NLL LM PAC #N LM SWAP
  8    DROP PAD +  SWAP CMOVE ;
  9  : >DA ' DAT  DA PAD + SWAP
 10    CMOVE ;
 11  : >VV NVV VM PAC #N VM SWAP
 12    DROP PAD + SWAP  CMOVE ;
 13  : >DD C>L PAD  #NR @ #INDEX
 14    RECLEN CMOVE UPDATE ;
 15  : LCOR >LL >VV >DA  >DD ; -->

SCR # 197
  0  ( BUSI PRINTING cntd         ef)
  1 : @PD PADD CNT# @ + 4 + @ C>L
  2  #SUCH IF MES2 0 ELSE @MEM .PD 1
  3  THEN ;
  4 : (?MW) ( -F) PAD 24 + C@ 49 = ;
  5  : ?MW ( -N) (?MW) IF MW1 ELSE
  6   MW2 THEN ;
  7 : NETB ( -D) P>Z ?MW NET 2DUP
  8   .NE ;
  9 : SUM ( D-D') 2 @MM DUP ANZ +!
 10   UDN* 2DUP .SU ;
 11 : !SUM ( D) (?MW) IF 1MW 2+!
 12   ELSE 2MW 2+! THEN ;
 13   -->
 14
 15

SCR # 198
  0  ( BUSI AUSGABE SONDEREIN.   ef)
  1  : SP>Z 23 SA 7 N>Z ;
  2  : .SM 0 @MM ABS 0 #N 18 4
  3    (RADJ) ;
  4  : .SB 2 SA 20 24 TEX> ;
  5  : .SP 23 SA 7 N>Z #DM 49 8
  6    (RADJ) ;
  7 : ?SMW ( -F) 22 SA C@ 49 = ;
  8 : SNET ( -D) SP>Z ?SMW IF MW1
  9   ELSE MW2 THEN NET 2DUP .NE ;
 10 : SSUM ( D-D') 0 @MM ABS DUP
 11   ANZ +! UDN* 2DUP .SU ;
 12 : !SSUM ( D) ?SMW IF 1MW ELSE
 13   2MW THEN 2+! ;
 14        -->
 15
```

```
SCR # 199
  0 ( ZEILE DRUCKEN                  EF)
  1 : (.MW) ( F) IF 49 ELSE 50 THEN
  2   75 ADJ EMIT ;
  3 : .MW (?MW) (.MW) ;
  4 : >Z .BN .OM .LM @PD AND IF
  5  NETB SUM !SUM .MW LCOR THEN ;
  6 : .SMW ?SMW (.MW) ;
  7 : >SZ .SM .SB .SP SNET SSUM
  8   !SSUM .SMW ;
  9
 10 : .ZS ZS 2@ .SU ;
 11
 12 : .RA 48 ADJ RAB @ 0 #N ATYPE
 13   80 BLOCK 12 + 10 ATYPE .ZS ;
 14
 15 -->

SCR # 200
  0  ( BUSI RECHNUNG RABATT       ef)
  1 ADRESSEN DEFINITIONS
  2 : ARA ANZ @ DUP 10 > IF 40 RAB !
  3   DROP   ELSE  5 > IF 33 RAB !
  4    ELSE 25 RAB ! THEN THEN ;
  5 : (R) ( D-D') RAB @ UDN* 100
  6    M/MOD  ROT 49 > IF 1. D+
  7    THEN 2DUP ZS 2+! ;
  8 : (RA) 2@ 2DUP (R) DMINUS D+ ;
  9 : 1RAB 1MW (RA) 1MW 2! ;
 10 : 2RAB 2MW (RA) 2MW 2! ;
 11 : ?RA PADD 19 + C@ DUP 78 = NOT
 12  IF 65 = IF ARA ELSE PADD 20 + @
 13  RAB ! THEN 0. ZS 2! 1RAB 2RAB
 14  .RA ELSE DROP THEN ;
 15  -->

SCR # 201
  0 -->
  1

SCR # 202
  0 ( SUMMENZEILEN                  EF)
  1 ADRESSEN DEFINITIONS
  2 : MW.  1000 - 0 #% 57 5 (RADJ) ;
  3
  4 : 1EZ ?TX IF 1MW 2@ MW1 BRT 2DUP
  5   1MW 2@ DMINUS D+ #DM 47 8
  6 (RADJ) MW1 MW. 1MW 2@ 2SWAP 1MW
  7 2! ELSE 1MW 2@ THEN #DM 64 8
  8  (RADJ) ;
  9
 10 : 2EZ ?TX IF 2MW 2@ MW2 BRT 2DUP
```

```
 11  2MW 2@ DMINUS D+ #DM 47 8
 12 (RADJ) MW2 MW. 2MW 2@ 2SWAP 2MW
 13 2! ELSE 2MW 2@ THEN #DM 64 8
 14  (RADJ) ;
 15 -->

SCR # 203
  0 ( SUMMENZEILEN                 EF)
  1
  2 : 3EZ PADD 25 + @ 0 2DUP 1MW 2+!
  3   ?TX IF 2DUP
  4   2DUP MW1 NET DMINUS D+ #DM 47
  5   8 (RADJ) MW1 MW. MW1 NET THEN
  6   #DM 64 8 (RADJ) ;
  7
  8
  9
 10
 11
 12 : 4EZ 1MW 2@ 2MW 2@ D+ #DM
 13   58 8 (RADJ) ;
 14  -->
 15

SCR # 204
  0 ( BANKVERBINDUNGEN DRUCKEN   EF)
  1 : 5EZ START 1+ BLOCK 45 0 TEX> ;
  2  : 6EZ START 1+ BLOCK RECLEN +
  3    45 0 TEX> ;
  4 : SU. 42 V @ - VE 1EZ 2 VE 2EZ
  5   2 VE 5EZ 3EZ 1 VE 6EZ 4EZ ;
  6
  7 : 1PZ VN 0 PTEX> ;
  8 : 2PZ ST 0 PTEX> ;
  9 : 3PZ PLZ 0 PTEX> ORT 8 PTEX> ;
 10 : 4PZ VN 30 PTEX> ;
 11 : 5PZ CO 30 PTEX> ;
 12 : 6PZ ST 30 PTEX> ;
 13 : 7PZ PLZ 30 PTEX> ORT 38
 14   PTEX> ;
 15  -->

SCR # 205
  0 ( AUFKLEBER DRUCKEN            EF)
  1 : AU. C>A 1 @MEM 3 VE 1PZ 2 VE
  2   2PZ 2 VE 3PZ PADD @ @MEM
  3   62 V @ - VE 4PZ
  4   2 VE 5PZ 2 VE 6PZ 4 VE 7PZ ;
  5
  6 -->
  7
  8
  9
```

```
SCR # 206
  0  ( BUSI RECHN.SCHREIBEN cndt ef)
  1
  2
  3  : 1RS 0. ZS 2! 0 ANZ !
  4    0. 1MW 2! 0. 2MW 2!
  5  30 CNT# ! 0 H ! 0 V ! C>A @OF ;
  6
  7 : RS 1RS .HE 1 VE .AD 5 VE
  8   BEGIN PADD
  9   CNT# @ + @  DUP -100 = NOT
 10  WHILE   0< IF >SZ 30 ELSE >Z 6
 11  THEN CNT#  +! 1 VE
 12  REPEAT DROP ?RA C>A SU. AU. ;
 13
 14  -->
 15

SCR # 207
  0  ( BUSI PRINTING END          ef)
  1  : 1RE RS
  2    @I# 1+ I# ! UPDATE FLUSH
  3    75 V @ - VE ;
  4
  5  : BSTF 37 BLNR ! 1 BEST !
  6    C>A CLR .CM1 WAS ;
  7
  8  : CONT 1 BLNR +! 1 BEST +!
  9    C>A CLR .CM1 WAS  ;
 10  : RESS 37 BLNR ! PRON BEST @ 0
 11    DO  1RE 1 BLNR +! LOOP
 12    PROF .INV ;
 13
 14  : RECHNUNG FLUSH C>L D<S NQ
 15    C>A .INV ; ;S

SCR # 208
  0 ( ALLNEW FORTH                EF)
  1 FORTH DEFINITIONS
  2 : ALLNEW EMPTY-BUFFERS CLR 3 2
  3   CU ." PROGRAMM-DISKETTE ENTNEH
  4 MEN" 4 2 CU ." DATA INIT DISK EI
  5 NLEGEN" 5 2 CU ." RETURN TASTE D
  6 RUECKEN" KEY DROP 80 BLOCK
  7 UPDATE DROP 319 BLOCK UPDATE
  8 DROP 318 BLOCK UPDATE DROP
  9 10 2 CU ." FORMATIERTE DISKETTE
 10  EINLEGEN" 11 2 CU ." RETURN TAS
 11 TE DRUECKEN" KEY DROP
 12 ADRESSEN C>A 1 FIRST# ! UPDATE
 13 ADRESSEN C>L 1 FIRST# ! UPDATE
 14
 15 FORTH FLUSH ADRESSEN RUN ; -->
```

```
SCR # 209
  0  ( BUSI STARTUP                ef)
  1
  2  : STARTUP CLR 3 2 CU
  3  ." BUSI-PACK " 5 2 CU
  4  ." (C) 1984 BY ING W. HOFACKER
  5 GMBH"
  6  7 4 CU ." ALLNEW"
  7  8 4 CU ." ADRESSEN RUN "
  8  9 4 CU ." LAGER RUN"
  9 10 2 CU ." DATEN DISKETTE EINLEG
 10 EN "
 11  ADRESSEN C>A FORTH  WAS ;
 12
 13
 14
 15   ;S
```

Notizen

Bezeichnungen:

n, n1	16 bit Zahlen mit Vorzeichen
d, d1	32 bit Zahlen mit Vorzeichen
u, u1	16 bit Zahl ohne Vorzeichen
addr	Adresse
b	8 bit Byte
c	7 bit ASCII-Zeichen
f	Bool'sche Zahl

Bei der Angabe der Veränderungen des Stapels ist links der Zustand des Stapels vor dem Aufruf des Wortes, rechts der Zustand des Stapels nach dem Aufruf des Wortes angegeben. Das in dieser Bezeichnung am weitesten rechts stehende Element ist oberstes Element des Stapels.

STAPELÄNDERUNGEN:

DUP	(n --> n n)	Verdoppeln der obersten Zahl
DROP	(n -->)	Löschen der obersten Zahl
SWAP	(n1 n2 --> n2 n1)	Vertauschen der beiden obersten Zahlen
OVER	(n1 n2 --> n1 n2 n1)	Zweite Zahl des Stapels im TOS kopieren
ROT	(n1 n2 n3 --> n2 n3 n1)	Zyklisches Vertauschen der obersten drei Zahlen
-DUP	(n --> n ?)	Verdoppeln der Zahl, nur wenn sie nicht Null ist
>R	(n --> n ?)	Oberste Zahl auf den Return-Stapel schreiben
R>	(--> n)	Oberste Zahl des Return-Stapels auf den Stapel holen
R	(--> n)	Oberste Zahl des Return-Stapels auf den Stapel kopieren

ARITHMETISCHE UND LOGISCHE BEFEHLE:

+	(n1 n2 --> sum)	Addieren
D+	(d1 d2 --> sum)	Addieren doppelt langer Zahlen
–	(n1 n2 --> diff)	Subtrahieren (n1 – n2)
*	(n1 n2 --> prod)	Multiplizieren
/	(n1 n2 --> quot)	Dividieren (n1 / n2)
MOD	(n1 n2 --> rem)	Rest der Division (Modulo)
/MOD	(n1 n2 --> rem quot)	Quotient und Rest der Division
*/MOD	(n1 n2 n3 --> rem quot)	Berechnung von n1 * n2 / n3 mit doppelt langen Zwischenergebnis
*/	(n1 n2 n3 --> quot)	Wie */MOD, nur Quotient auf dem Stapel
MAX	(n1 n2 --> max)	Max ist die größere der beiden Zahlen n1, n2
MIN	(n1 n2 --> min)	Min ist die kleiner der beiden Zahlen n1, n2
ABS	(n --> absolute)	Absolutwert einer 16 bit Zahl
DABS	(d --> absolute)	Absolutwert einer 32 bit Zahl
MINUS	(n --> -n)	16 bit Zahl wird negativ
DMINUS	(d --> -d)	32 bit Zahl wird negativ
AND	(n1 n2 --> and)	Logisches UND, bitweise
OR	(n1 n2 --> or)	Logisches ODER, bitweise
XOR	(n1 n2 --> xor)	Logisches EXCLUSIV ODER, bitweise

STEUERWORTE:

DO . . . LOOP		
DO	(end+1 start -->)	Setzt Schleifenparameter
LOOP	(-->)	Erhöht Zählparameter um Eins
I	(--> n)	Holt Zählparameter auf Stapel
LEAVE	(-->)	Verlässt die Schleife beim nächsten LOOP oder +LOOP
DO . . . +LOOP		Wie DO . . . LOOP, aber addiert n zu Zählparameter
+LOOP	(n -->)	
IF . . . ENDIF	(f -->)	Programm zwischen IF und ENDIF (THEN) wird ausgeführt,
IF . . . THEN	(f -->)	wenn der Wert im TOS ungleich Null ist
IF . . . ELSE . . . ENDIF	(f -->)	Programm zwischen IF und ELSE wird ausgeführt, wenn Wert
IF . . . ELSE . . . THEN		im TOS ungleich Null ist, sonst wird Programm zwischen ELSE und ENDIF (THEN) ausgeführt.
BEGIN . . . UNTIL	(f -->)	Programm zwischen BEGIN und UNTIL (END) wird solange aus-
BEGIN . . . END	(f -->)	geführt, bis im RPS vor UNTIL (END) ein Wort ungleich Null ist
BEGIN . . . WHILE		Solange der TOS vor WHILE ungleich Null ist, wird Programm
. . . REPEAT	(f -->)	zwischen BEGIN und REPEAT ausgeführt. Mit TOS gleich Null vor WHILE wird Schleife verlassen.

SPEICHERVERÄNDERUNG:

@	(addr --> n)	Holen einer Zahl von Adresse addr
!	(n addr -->)	Speichern einer Zahl nach Adresse addr
C@	(addr --> b)	Holen eines Byte von Adresse addr
C!	(b addr -->)	Speichern eines Byte nach Adresse addr
?	(addr -->)	Inhalt einer Variablen ausdrucken
+!	(n addr -->)	Wert einer Variablen um n erhöhen
CMOVE	(from to u -->)	u Byte im Speicher verschieben
FILL	(addr u b -->)	u Byte ab Adresse addr mit b füllen
ERASE	(addr u -->)	u Byte ab Adresse addr mit 00 füllen
BLANKS	(addr u -->)	u Byte ab Adresse addr mit Leerzeichen füllen

VERGLEICHSBEFEHLE:

<	(n1 n2 --> f)	f = 1, wenn n1 < n2
>	(n1 n2 --> f)	f = 1, wenn n1 > n2
=	(n1 n2 --> f)	f = 1, wenn n1 = n2
0<	(n --> f)	f = 1, wenn n kleiner Null
0=	(n --> f)	f = 1, wenn n gleich Null
U<	(Z1 u2 --> f)	f = 1, wenn U1 und U2.
U<	(u1 u2 --> f)	f = 1, wenn u1 < u2

ZAHLENSYSTEME:

DECIMAL	(-->)	Dezimalrechnung
HEX	(-->)	Hexadezimalrechnung
BASE	(--> addr)	Variable, enthält Zahlenbasis

DEFINITIONSWORTE:

: xxx	(-->)	Beginn der Definition von xxx
;	(-->)	Ende der Definition
VARIABLE xxx	(n -->) xxx: (--> addr)	Definition einer Variablen xxx mit Wert n
CONSTANT xxx	(n -->) xxx: (--> addr)	Definition einer Konstanten xxx mit Wert n
‹BUILDS . . . DOES›	does: (--> addr)	Definiert neue Compiler Worte mit unterschiedlichen Compile und Laufzeitverhalten
CREATE xxx	(-->)	Trägt xxx ins Wörterbuch ein, mit HERE als Codefeldaresse. Bit 8 der Namenlänge gleich Eins
SMUDGE		Ändert Bit 8 der Namenlänge in obersten Eintrag

EIN- UND AUSGABE:

.	(n -->)	Druckt Wort des TOS aus
R	(n fieldwidth -->)	Druckt rechtsbündig (Feldbreite im TOS) aus
D.	(d -->)	Druckt doppelt lange Zahlen aus
D.R	(d fieldwidth -->)	Druckt doppelt lange Zahlen rechtsbündung aus
U.	(u -->)	Druckt TOS als Zahl ohne Vorzeichen aus
CR	(-->)	Gibt ein Carriage Return aus
SPACE	(-->)	Gibt ein Leerzeichen aus
SPACES	(n -->)	Gibt ein Leerzeichen aus
." xxx "	(-->)	Gibt Text xxx aus
TYPE	(addr u -->)	Gibt u Zeichen, gespeichert ab Adresse addr aus
COUNT	(addr --> addr+1 u)	Ändert Längenbyte für TYPE Ausgabe
?TERIMINAL	(--> f)	f = 1, wenn eine Taste gedrückt
KEY	(--> c)	Eingabe von Zeichen vom Tastenfeld
EMIT	(c -->)	Ausgabe des ASCII-Zeichen C
EXPECT	(addr n -->)	Erwartet n Byte Eingabe; speichert ab Adresse addr
WORD	(c -->)	Liest ein Wort, begrenzt durch c, aus Eingabespeicher

EIN- AUSGABEFORMATIERUNG:

NUMBER	(addr --> d)	Setzt eine Zeichenkette ab Adresse addr in eine doppelt lange Zahl um
‹#	(-->)	Beginn der Zeichenformatierung
#	(d --> d)	Wandle eine Zahlenstelle in ASCII-Zeichen

#S	(d --> 0 0)	Wandle restliche Stelle in ASCII-Zeichen
SIGN	(n d --> d)	Setze Vorzeichen ein
#>	(d --> addr u)	Beende Formatierung
HOLD	(c -->)	Füge ASCII-Zeichen c in Zeichenkette ein

WÖRTERBUCH:

CONTEXT	(--> addr)	Bringt Adresse des CONTEXT Wörterbuches
CURRENT	(--> addr)	Bringt Adresse des CURRENT Wörterbuches
FORTH	(-->)	Setzt CONTEXT und CURRENT auf FORTH Wörterbuch
DEFINITIONS	(-->)	Setzt CURRENT auf CONTEXT
VOCABULARY xxx	(-->)	Erzeugt Wörterbuch xxx
VLIST	(-->)	Listet Inhalt des Wörterbuches
FENCE	(--> addr)	Systemvariable, enthält die Adresse des obersten geschützen Eintrags im Wörterbuch

SYSTEM WORTE:

(	(-->)	Beginn eines Kommentars, Kommantierende ist)
FORGET xxx	(-->)	Löschen eines und aller späteren Einträge im Wörterbuch
ABORT	(-->)	Fehlerabbruch
' xxx	(--> addr)	Suche Codefeldadresse von xxx
HERE	(--> addr)	Adresse der nächsten freien Stelle im Wörterbuch
PAD	(--> addr)	Adresse von PAD
IN	(--> n)	Offset zum Eingabespeicher
SP@	(--> addr)	Adresse des TOS
ALLOT	(n -->)	**n Bytes im Wörterbuch reservieren**
	(n -->)	**Speicher von n an Adrese HERE**
C,	(b -->)	**Speicher n des Byte b and Adresse HIRE**

DISKETTENSPEICHERUNG:

LIST	(n -->)	Textfeld (Screen) n wird ausgegeben
LOAD	(n -->)	Textfeld (Screen) n wird compiliert
BLOCK	(n --> addr)	Diskettenblock n wird eingelesen, Anfangsadresse im Speicher auf Stapel
FLUSH	(-->)	Geänderte Blöcke werden auf Diskette geschrieben.
UPDATE	(-->)	Augenblicklichen Block als geändert markieren
EMPTY-BUFFERS	(-->)	Alle Blöcke im Speicher löschen
BLK	(--> addr)	Variable, enthält augenblickliche Blocknummer
B/BUF	(--> n)	Constante, Blockgröße in Byte
SCR	(--> addr)	Variable, enthält augenblickliche Textfeldnummer

Literaturverzeichnis

1. Ronald Knuth: The Art of Computer Programming
   Vol. 3: Sorting and Searching.
   Verlag Addison Wesley

2. Zeitschrift FORTH DIMENSIONS
   FORTH Interest Group, Po. Box 1105, San Carlos Ca 9407

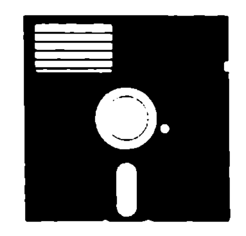
3. M. Derik & L. Baker: FORTH Encyclopedia
   Mountain View Press, Mountain View CA

4. E. Denert, R. Frank: Datenstrukturen
   B. i. Wissenschaftsverlag

5. E. Flögel: FORTH-Handbuch
   Hofacker Verlag, Holzkirchen

6. Ronald Zech: Die Programmiersprache FORTH
   Franzis Verlag, München

Alle Programme aus diesem Buche auf Diskette



Für denjenigen, der nicht die Zeit und Muse hat, die Programme einzugeben, halten wir eine Diskette bereit, die alle Programme aus diesem Buche enthalten.

Wir senden Ihnen diese gerne gegen Vorauszahlung von DM 299,– incl. Versand und Verpackung zu.

Verwenden Sie dieses Blatt als Bestellschein.

. . . .
Name

. . . .
Vorname

. . . .
Straße

. . . .
PLZ Ort

☐ Ich bestelle die Diskette, voll mit Programmen zum Buch #200 zu DM 299,– incl. Porto und Verpackung. Für folgenden Rechnertyp:

- ☐ Commodore 64
- ☐ ATARI 800/800XL (Bitte unbedingt ankreuzen !)
- ☐ APPLE II, IIe 48K

Dazu wird ein Fig-FORTH für den entsprechenden Rechnertyp benötigt (siehe auch folgende Seite) !

☐ Den Betrag von DM 299,– habe ich heute auf Ihr Postscheck-Kto. München 15994–807 überwiesen.

☐ Bitte liefern Sie per Nachnahme. Hier kommen noch NN-Gebühren in Höhe von 6,50 DM hinzu.

. . . .
Datum Unterschrift (f. Jugendliche unter 18 Jahr der Erziehungsberechtigte)

Notizen

FORTH

ELCOMP FORTH ist eine erweiterte Implementierung der Fig-FORTH Release 1.1. Der Befehlsumfang des Kernels entspricht dem Wortevorrat, wie er in der FORTH Encyclopedia von M. Derik und d. Baker angegeben ist. Für einzelne Rechner sind Utilities, Grafik, usw. als Textfelder vorhanden. Zum Erstellen von Textfeldern steht ein zeichenorientierter Editor zur Verfügung.

ELCOMP	**FORTH für APPLE** Utilities, Editor, Hi- und Lo RES Grafik	Best.-Nr. 6155	129,– DM
ELCOMP	**FORTH für ATARI** Utilities, Editor, Grafik, Ton, Player Missle	Best.-Nr. 7055	199,– DM
	Floating Point (nur für ATARI)	Best.-Nr. 7230	98,– DM
ELCOMP	**FORTH für Commodore 64** Utilities, Editor, Assembler	Best.-Nr. 4960	299,– DM
	FORTH für TRS-80/GENIE Editor, Assembler	Best.-Nr. 5026	199,– DM

In allen FORTH-Versionen ist das FORTH Handbuch Nr. 137 inbegriffen.



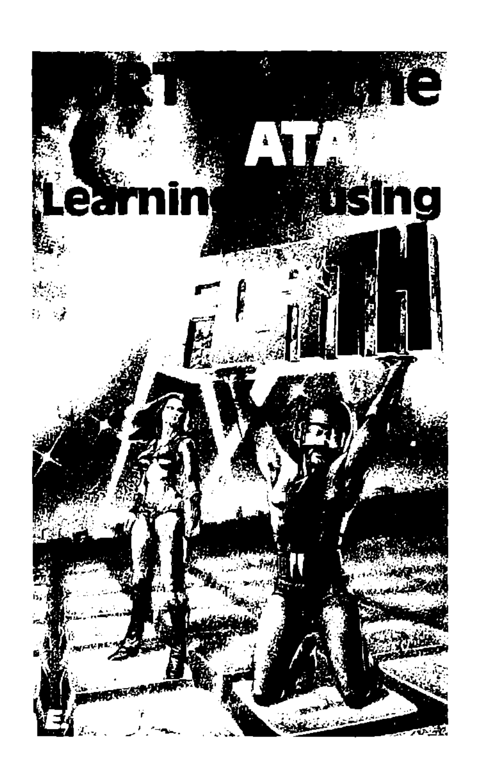
FORTH Handbuch

von E. Flögel

Erstmals wurde hier in deutscher Sprache, in einem umfassenden Werk, die Grundlagen dieses so leistungsfähigen Interpreters/Compilers für den Einsteiger verständlich dargestellt. Viele Beispiele erleichtern das Verständnis und die Anwendung, welche sich von der einfachen Steuerung bis hin zu Adressen- u. Dateiverwaltungen erstreckt.

FORTH gibt es heute für fast alle Personalcomputer: ATARI, APPLE, Commodore 64, GENIE, TRS-80, usw. FORTH-Programme sind transportabel ! (189 Seiten)

Best.-Nr. 137 49,– DM



FORTH on the ATARI – Learning by Using

von E. Flögel

Dies ist das erste und zur Zeit einzige Buch auf dem Weltmarkt, welches die Programmiersprache FORTH speziell auf Ihren ATARI-Personalcomputer abhandelt. Einführung, Grundelemente v. FORTH, Strukturen, Programmbeispiele, Ton und Grafik, Ein- und Ausgabeprogrammierung und eine komplette Adressenverwaltung in FORTH. (118 Seiten, englisch)

Best.-Nr. 170 29,80 DM

Weitere interessante Bücher von Hofacker:

Best.-Nr.	Titel	Preis / DM
	BÜCHER in deutscher Sprache aus dem HOFACKER-Verlag	
1	Transistor Berechnungs- und Bauanleitungsbuch – 1	29,80
2	Transistor Berechnungs- und Bauanleitungsbuch – 2	19,80
3	Elektronik im Auto	9,80
4	IC-Handbuch, TTL, CMOS, Linear	19,80
5	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	IC-Schaltungen, TTL, CMOS, Linear	19,80
7	Elektronik Schaltungen	19,80
8	IC-Bauanleitungs-Handbuch	19,80
9	Feldeffektransistoren	9,80
10	Elektronik und Radio	19,80
11	IC-NF Verstärker (i. V.)	9,80
12	Beispiele integrierter Schaltungen (BIS)	19,80
13	HEH, Hobby Elektronik Handbuch	9,80
15	Optoelektronik Handbuch	19,80
16	CMOS Teil 1, Einführung, Entwurf, Schaltbeispiele	19,80
17	CMOS Teil 2, Entwurf und Schaltbeispiele	19,80
18	CMOS Teil 3, Entwurf und Schaltbeispiele	19,80
19	IC-Experimentier Handbuch	19,80
20	Operationsverstärker	19,80
21	Digitaltechnik Grundkurs	19,80
22	Mikroprozessoren, Eigenschaften und Aufbau	19,80
23	Elektronik Grundkurs, Kurzlehrgang Elektronik	9,80
24	Progr. in Maschinensprache mit Z80, II	29,80
25	65000 Microcomputer Einführung (i. V.)	39,00
26	Mikroprozessor, Teil 2	19,80
27	BASIC-M Anwender-HB f. 6800/09/68000 (Motorola)	29,80
28	Lexikon + Wörterbuch f. Elektr. u. Mikroprozessor	29,80
29	Mikrocomputer Datenbuch	49,00
30	Floppy Disk Selbstbau-Handbuch (i. V.)	49,00
31	57 Programme in BASIC	39,00
32	ATARI BASIC, für Selbststudium und Praxis	39,00
33	Microcomputer Programmierbeispiele	19,80
34	TINY-BASIC Handbuch	19,80
35	Der freundliche Computer	29,80
101	Das große Druckerbuch (i. V.)	29,80
102	Statistik in BASIC (i. V.)	29,80
103	Oszillographen-Handbuch	19,80
104	Portable Computer Handbuch (i. V.)	39,00
108	Rund um den Spectrum (Progr., Tips und Tricks)	29,80
109	6502 Microcomputer Programmierung	29,80
110	Programmierhandbuch für PET	29,80
111	Programmieren mit TRS-80 (GENIE)	29,80
112	PASCAL-Programmier-Handbuch	29,80
113	BASIC-Programmier-Handbuch (mit BASIC-Kurs)	19,80
114	Der Microcomputer im Kleinbetrieb	39,80
115	6809 Programmier Handbuch (i. V.)	49,00
116	Einführung 16-Bit Microcomputer	29,80
118	Programmieren in Maschinensprache mit dem 6502	49,00
119	Programmieren in Maschinensprache (Z80), I	39,00
120	Anwenderprogramme für TRS-80 und GENIE	29,80
121	Microsoft BASIC-Handbuch	29,80
122	BASIC für Fortgeschrittene	39,00
123	IEC-Bus Handbuch	19,80
124	Progr. in Maschinensprache mit Commodore-64	29,80
127	Einführung i. d. Microcomputer-Progr. mit 6800	49,00
128	Programmieren mit dem CBM	29,80

Best.-Nr.	Titel	Preis / DM
130	Programmierbeispiele für CBM	19,80
132	CP/M-Handbuch	19,80
133	Handbuch für MS/DOS (i. V.)	29,80
136	Funktionsanalyse (i.V.)	79,00
137	FORTH – Grundlagen, Einführung, Beispiele	49,00
139	BASIC für blutige Laien (speziell f. TRS-80, Genie)	19,80
140	Progr. i. BASIC u. Maschinencode mit dem ZX81	29,80
141	Programme f. VC-20 (Spiele, Utilities, Erweiterungen)	29,80
143	35 Programme für den ZX81	29,80
144	33 Programme für den ZX-Spectrum	29,80
145	64 Programme für den Commodore-64	39,00
146	Hardware-Erweiterungen für den Commodore-64	39,00
147	Beherrschen Sie Ihren Commodore-64	19,80
148	Programmierhandbuch für SHARP	49,00
149	Programme für TI 99/4A	49,00
175	Astrologie auf dem ATARI 800	49,00
187	Mehr als 29 Programme für den Commodore 64	29,80
190	Das große Spielebuch für ATARI 600XL/800XL	29,80
200	FORTH-Anwendungen	49,00
8029	Z-80 Assembler-Handbuch	29,80
	BÜCHER in englischer Sprache von ELCOMP-Publishing, Inc., Los Angeles, CA.	
150	Care and Feeding of the Commodore PET	19,80
151	8K Microsoft BASIC Reference Manual	9,80
152	Expansion Handbook for 6502 and 6800	19,80
154	Complex Sound Generation Using the SN76477	9,80
156	Small Business Programs	29,80
158	The Second Book of Ohio Scientific	19,80
159	The Third Book of Ohio Scientific	29,80
160	The Fourth Book of Ohio Scientific	29,80
161	The Fifth Book of Ohio Scientific	19,80
162	ATARI Games in BASIC	19,80
163	The Peripheral Handbook (i. V.)	29,80
164	ATARI-BASIC Learning by Using	19,80
166	Programming in 6502 Machinelanguage PET/CBM	49,00
169	How the Progr. your ATARI in 6502 Machinelanguage	29,80
170	FORTH on the ATARI – Learning by Using	29,80
171	See the Future with your ATARI (Astrology)	49,00
172	Hackerbook I (Tricks + Tips for your ATARI)	29,80
173	PD-Program Descriptions (ATARI)	9,80
174	ZX-81/TIMEX Progr. i. BASIC a. Machine Lang.	29,80
176	Programs + Tricks for VIC's	29,80
177	CP/M – MBASIC and the OSBORNE	29,80
178	The APPLE in Your Hand	39,00
182	The Great Book of Games Vol. I - Games f. the C-64	29,80
183	More on the Sixtyfour (Commodore-64)	39,00
184	How to Progr. your C-64 i. 6502/10 Machinelang.	29,80
185	Commodore 64 Tune-up	39,00
186	Small Business Programs for the Commodore 64	49,00

Der HOFACKER Verlag produziert und vertreibt neben einer sehr großen Auswahl an Fachbüchern für Elektronik und Microcomputertechnik noch:

- Leerplatinen und Bauanleitungen für Zusatzeinrichtungen für Ihren Personalcomputer, sowie
- Programme (Software) und Leercassetten (C-10) für die bedeutenden Personalcomputer.

(i. V. bedeutet: Buch ist in Vorbereitung)



HOLZKIRCHEN SINGAPORE LOS ANGELES

ISBN 3-88963-200-9